ENTWICKLUNG VON SOFTWARE ZUM SCHNITT VON MPEG2-KODIERTEN VIDEODATEN

Diplomarbeit
im Fachbereich Photoingenieurwesen und Medientechnik
an der Fachhochschule Köln

Autor
Matthias Fokken
aus Köln
Matr.-Nr. 11032552

Referent: Prof. Dr. rer. nat. D. Kunz
Korreferent: Dr. H. Strauss, VCS AG Bochum

Köln, 10. Juli 2006

DEVELOPMENT OF SOFTWARE TO CUT MPEG2-CODED VIDEO DATA

Thesis
at the Department
of
Imaging Sciences and Media Technology
University of Applied Sciences Cologne

Author
Matthias Fokken
Cologne
Matr.-Nr. 11032552

First Reviewer: Prof. Dr. rer. nat. D. Kunz
Second Reviewer: Dr. H. Strauss, VCS AG Bochum

Cologne, July, 10th 2006

Kurzbeschreibung

Titel:

Entwicklung von Software zum Schnitt von MPEG2-kodierten Videodaten

Autor:

Matthias Fokken

Referenten:

Prof. Dr. rer. nat. D. Kunz, Dr. H. Strauss, VCS AG Bochum

Zusammenfassung:

Im Rahmen dieser Diplomarbeit wurde eine Software entwickelt, welche unter Verwendung der Intel-UMC-Klassen bildgenau MPEG2-Videoelementarströme nach einer gegebenen Schnittliste schneidet. Dazu wird der Videodatenstrom auf Bitstromebene unter Berücksichtigung der GOP´s mit ihren einzelnen Bildern und deren Bildtypen analysiert und verarbeitet. In Situationen, bei denen Referenzbilder für einzelne in den Ausgangsdatenstrom übertragene Bilder verloren gehen, wird eine Neukodierung der betroffenen Umgebung eingeleitet. In allen Fällen wurde sichergestellt, dass das Ausgabevideo konform zur MPEG2-Spezifikation ISO/IEC 12818-2 ist.

Stichwörter:

MPEG2, Videoschnitt, MPEG2-Elementarstrom, Schnittliste, Offline-Schnitt

Sperrvermerk:

Für die vorgelegte Arbeit gibt es keinen Sperrvermerk

Datum:

10. Juli 2006

Brief Description

Title:

Development of software to cut MPEG2-coded video data

Author:

Matthias Fokken

Advisores:

Prof. Dr. rer. nat. D. Kunz, Dr. H. Strauss, VCS AG Bochum

Abstract:

In this thesis, a software to cut MPEG2-coded video elementary streams according to a given cutting list has been developed using the Intel-UMC-classes. The software analyses the video bitstream and processes it taking into account the structure of the start/end GOP and the types of the involved pictures in order to obtain frame-correct cuts. If P and B frames to be transferred to the output stream refer to frames that are not contained in the output stream, these references are removed and then these frames are transcoded accordingly. In all cases, the output file is compliant to the MPEG2-specification ISO/IEC 12818-2.

Keywords:

MPEG2, video cut, elementary stream, edit decision list

Remark of closure:

This thesis is not closed.

Date:

July, 10th 2006

Danksagung

Ich möchte mich für die vielseitige Unterstützung bedanken, die ich während meiner Diplomarbeit erfahren habe.
Der VCS-Aktiengesellschaft danke ich für die Möglichkeit der Durchführung dieser Diplomarbeit, besonders bedanke ich mich bei Dr. Holger Strauss sowie bei allen anderen Kollegen, die mich unterstützt und für das gute Arbeitsklima gesorgt haben.
An der Fachhochschule Köln gilt mein Dank Herrn Prof. Dr. rer. nat. D. Kunz für die hervorragende Betreuung und das Interesse an meiner Arbeit.

Besonderer Dank gilt auch meiner Familie und meinen Freunden, sowie meiner lieben Freundin Marina für die Unterstützung und das Korrekturlesen.

Eidesstattliche Erklärung

Ich versichere hiermit, die vorgelegte Arbeit in dem gemeldeten Zeitraum ohne fremde Hilfe verfasst und mich keiner anderen als der angegebenen Hilfsmittel und Quellen bedient zu haben.

Köln, 10. Juli 2006

Sperrvermerk

Die vorgelegte Arbeit unterliegt keinem Sperrvermerk

Weitergabeerklärung

Ich erkläre hiermit mein Einverständnis, dass das vorliegende Exemplar meiner Diplomarbeit oder eine Kopie hiervon für wissenschaftliche Zwecke verwendet werden darf.

Köln, 10. Juli 2006

Inhaltsverzeichnis

1. Einführung

Derzeit entstehen bei der VCS Aktiengesellschaft in der Abteilung MBS[1] verschiedene MPEG-basierende[2] Digital-Video-Broadcast- und -editingtools. Dieses Kapitel stellt eine der zu entwickelnden Komponenten in ihrem Umfeld vor und begründet ihre Notwendigkeit. Mit der Planung und Entwicklung bis hin zum Ergebnis und der Diskussion des Ergebnisses beschäftigt sich diese Diplomarbeit.

1.1. Erläuterung der technischen Fragestellung

Innerhalb eines Digital-Video-Broadcast-Workflows hat man es häufig mit sehr großen Datenmengen zu tun. In Server-Client-Umgebungen sind zusätzlich die Bandbreiten beschränkt, sodass ein Online-Schnitt[3] kaum möglich ist.
So werden Lowres-Kopien[4] des Originalmaterials auf verschiedene Weise erzeugt und zur gegebenen Zeit kapazitätenschonend zu dem Client übertragen, an welchem der Schnitt vollzogen wird. Am Ende des Offline-Schnittes[5] steht im Gegensatz zum Online-Schnitt kein Ausgabevideo, sondern eine EDL[6], in welcher eine genaue Auflistung der gewünschten zu kopierenden Bereiche (im folgenden „Szene“ genannt) mit ihren In- und Outpoints festgehalten ist. Jede Szene zeigt zusätzlich auf das jeweilige Quellvideo. Anhand

[1] MBS = Media Broadcasting Solutions
[2] MPEG = Motion Picture Experts Group
[3] Online-Schnitt: der Schnitt am Originalmaterial, vgl. Offline-Schnitt
[4] Lowres = Low Resolution, niedrigauflösend
[5] Offline-Schnitt: der Schnitt an minderqualitativen Kopien
[6] EDL = Edit Decision List, Schnittliste

dieser EDL und den Highres-Videofiles[7] ist es möglich, das Ausgabevideo zu generieren.
Dazu wird die EDL zurück zum Server überspielt. Der Server hat nun die Aufgabe, diese EDL mit den ihm vorliegenden Highres-Videomaterial umzusetzen.
Es ist also eine Komponente notwendig, welche anhand einer EDL den kompletten Zusammenschnitt eines oder mehrerer Highres-MPEG-Videofiles als Quelle vollzieht. Dabei ist besonderes Augenmerk auf die Gültigkeit der MPEG-Ausgabevideos zu legen. Beim Zusammenschnitt des Ausgabevideos sind die Schnittpunkte aus der EDL bildgenau umzusetzen. Die Darstellungsqualität des Zielvideos ist auf dem Niveau der Quellvideos zu halten. Der Zeit- bzw. Rechenaufwand soll so gering wie möglich gehalten werden.

1.2. Abgrenzung der Diplomarbeit

Im Rahmen dieser Diplomarbeit entsteht eine Serveranwendung, welche mittels einer EDL und verschiedenen Quellvideos als Eingabe ein fertig geschnittenes der EDL entsprechendes Video erstellt. Der Schnitt bezieht sich ausschließlich auf Videomaterial, der Schnitt von Audiomaterial ist nicht Bestandteil der zu entwickelnden Komponente und somit der Diplomarbeit. Als Quell-Videoformat werden alle im MPEG-2-Standard vorkommenden Konzepte verarbeitet werden können. Die aus der EDL übernommenen Schnittpunkte werden bildgenau umgesetzt. Dabei werden ausschließlich harte Schnitte unterstützt. Das Laufzeitverhalten wird nach technischen Möglichkeiten optimiert.

[7] Highres = High Resolution, hochauflösend

2. Theorie

2.1. Aufbau von Video-Elementarströmen nach MPEG-Spezifikation

Ziel dieses Projektes ist es, MPEG2-konforme Video-Elemetarströme bildgenau zu schneiden. Dazu ist es unumgänglich, sich sehr detailliert mit dem Aufbau eines MPEG2-Elementarstroms zu beschäftigen.

Abbildung 2.1 zeigt den Aufbau eines MPEG-Videos. Nach der MPEG2-Spezifikation ISO/IEC 13818-2[1] wird das Grundgerüst gebildet durch die hierarchisch aufgebaute Layer-Struktur.

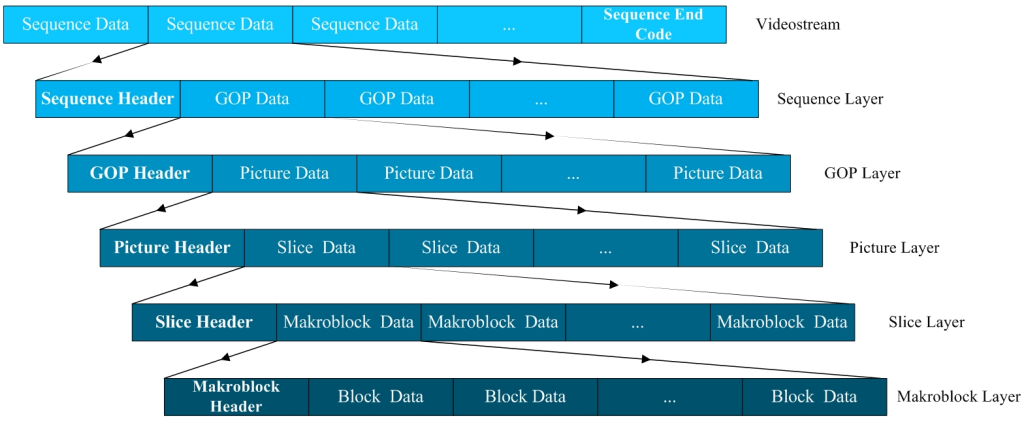

Abbildung 2.1.: Layer-Struktur von MPEG-Video

Der *Sequence-Layer* wird gebildet aus ein oder mehreren *GOP´s*[1], dem *Sequence-Header*, welcher jede Sequenz einleitet und den am Ende stehenden *Sequence-End-Code*.

[1] GOP = Group Of Pictures, Bildgruppe

Der in der Hierarchie folgende *GOP-Layer* fügt Bilder zu Bildgruppen zusammen – besteht also aus den einzelnen *Pictures*. Jeder *GOP* wird eingeleitet durch den *GOP-Header*.
Im als nächstes folgenden *Picture-Layer* steht das jeweils einzelne Bild als eines der charakteristischen Bildtypen, der I-, P- oder B-Bilder[2]. Jedes Bild wird angeführt von einem *Picture-Header*.
Anschließend folgen die nächsten Layer, der *Slice-Layer* und der *Macroblock-Layer*.

Im Folgenden werden die Eigenschaften der für diese Diplomarbeit relevanten Layer näher erläutert.

2.1.1. Die Sequenz und der *Sequence-Header*

Kodierte MPEG2-Datenströme besitzen Sequenzen und *Sequence-Header* mit folgenden Eigenschaften[1]:

- Jede Sequenz beginnt mit dem *Sequence-Header* optional enthaltend die *Sequence-Header-Extension*.
- *Sequence-Header* kehren optional in regelmäßigen Abständen wieder. Wiederholende *Sequence-Header* ermöglichen dem Dekoder den Einstieg in den Stream auch an anderen Stellen als dem Anfang. Dies ist zum Beispiel bei Systemen wie DVB-S, -C und -T[3] notwendig.
- Wenn *Sequence-Header* wiederholt werden, sollten deren Parameter die gleichen Werte wie im ersten *Sequence-Header* besitzen. Ebenso sollte, wenn die *Sequence-Header-Extension* beim ersten *Sequence-Header* vorkommt, sie auch bei allen weiteren *Sequence-Headern* vorkommen und die gleichen Werte haben.
- Es darf sich ein wiederholender *Sequence-Header* unmittelbar vor einem kodierten I- oder P-Bild befinden, bei Halbbildmaterial hat der *Sequence-Header* nur vor dem ersten Halbbild zu stehen.

[2]Bildkodierungstypen, vgl. Kapitel 2.1.3
[3]DVB = Digital Video Broadcasting, -S = Satellit, -C = Kabel, -T = Terrestrisch

Die wichtigsten Codes zur Behandlung des Datenstromes auf der *Sequence-Layer*-Ebene stellt Tabelle 2.1 dar.

Startcode	Hex	Binär
Sequence-Header	000001B3	00000000 00000000 00000001 10110011
Extension-Start	000001B5	00000000 00000000 00000001 10110101
Sequence-End	000001B7	00000000 00000000 00000001 10110111

Tabelle 2.1.: Relevante Codes im Sequence-Layer

Sequence-Header Der *Sequence-Header* enthält folgende Parameter:

- Die Parameter *Horizontal & Vertical Size* definieren die horizontale und vertikale Bildgröße der Luminanzkomponente.
- *Aspect Ratio* definiert das Bild-Seitenverhältnis der Sequenz.
- *Framerate* gibt Auskunft über die Bildwiederholfrequenz.
- Der Parameter *Bitrate* lässt erkennen, welche Bitrate der Datenstrom besitzt.
- Mit dem Parameter *VBV-BufferSize*[4] erhält man die Angabe über den notwendigen Buffer zum Dekodieren dieser Sequenz.
- Das folgende *Load-Intra-Quantisier-Matrix*-Bit gibt Auskunft über die Verwendung einer eigenen Quantisierer-Matrix, welche im Anschluss an dieses Flag übertragen werden würde. Ist dieses Flag Null, wird die Standard-Matrix benutzt.

Extension-Start Der *Extension-Startcode* leitet weitere Zusatzinformationen ein. Nach ihm sind neun weitere Typen von Zusatzinformationen möglich, jedes einzelne durch einen Startcode identifizierbar. Der Ort des Auftretens der einzelnen Zusatzinformationen ist abhängig vom Typ der Zusatzinformation. In Tabelle 2.2 stehen die für diese Arbeit relevanten Erweiterungen.

Sequence-End Der *Sequence-End-Code* steht für sich alleine und bildet das Ende des Datenstromes. Dieser Code veranlasst den Dekoder, den Dekodiervorgang zu beenden.

[4] VBV = Video Buffering Verifier

Name	Extension-Startcode-Identifier
Sequence-Extension-ID	0001
Sequence-Display-Extension-ID	0010
Picture-Coding-Extension-ID	1000

Tabelle 2.2.: Relevante Extension-Startcodes im Sequence-Layer

Sequence-Extension Die *Sequence-Extension* sollte sich immer unmittelbar nach dem *Sequence-Header* befinden. Es folgen nun die Parameter, die sich in der *Sequence-Extension* befinden.

- Die *Profile and Level ID* gibt Auskunft über den Bereich der verwendeten Untermenge an möglichen Werten einzelner Parameter. Durch die Verwendung solcher Parameter kann die Anforderung an die Komplexität des Dekoders herabgesetzt werden.
- *Progressive Sequence* gibt Auskunft, ob es sich um eine progressive Sequenz handelt. Ist dieses Flag Null, kann entweder eine frame- oder fieldpicture-basierte Sequenz vorliegen (vgl. 2.1.3).
- *Chroma-Format* beschreibt das verwendete Farbformat. Mögliche Werte sind 4:2:0, 4:2:2 und 4:4:4.
- Der Parameter *Low Delay* beschreibt, ob die Sequenz B-Bilder enthält. Ist dies nicht der Fall, kann auf eine Bildrücksortierung verzichtet werden (vgl. Kapitel 2.1.3).

Sequence-Display-Extension Die *Sequence-Display-Extension* befindet sich zwischen dem *Sequence-Header* und dem nächsten *Picture-Header*. Sie enthält keine den Dekodierungsprozess beeinflussenden Parameter, dennoch sollen hier die wichtigsten vorgestellt werden.

- Mit dem Parameter *Video Format* wird angegeben, um welches Videoformat es sich handelt. Mögliche Werte sind PAL, NTSC, SECAM, usw.
- *Color Primaries* beschreibt die Farbkoordinaten innerhalb des Quellvideomaterials, *Transfer Characteristics* die optoelektronische Transfercharakteristik der Bilder im Quellvideo und *Matrix Coefficients* die Koeffizienten, welche benutzt werden, sollte das

Luminanz- und Chrominanzsignal von den Grundfarben Grün, Blau und Rot abweichen.

- Die Werte *Display Horizontal & Vertical Size* bestimmen ein Rechteck, welches den vom Player darzustellenden Bereich des Videos markiert. Diese Werte beeinflussen den Display-Prozess des verwendeten Players.

Picture-Coding-Extension Wird behandelt in Kapitel 2.1.3.

2.1.2. Der *GOP* und der *GOP-Header*

Die *GOP´s* und die *GOP-Header* besitzen folgende Eigenschaften[1]:

- *GOP´s* vereinen eine variable Anzahl von Bildern zu einer Gruppe, jede Gruppe muss mit einem I-Bild beginnen.
- *GOP´s* ermöglichen dem Dekoder den wahlfreien Zugriff (schneller Vor- bzw. Rücklauf an I-Bildern).
- Es gibt zwei Typen von *GOP´s*: Offene und Geschlossene. *Geschlossene GOP´s* treffen im Gegensatz zu den *Offenen GOP´s* keinerlei Bildvorhersagen aus vorherigen *GOP´s*. Mehr dazu im Kapitel 2.1.3.

Startcode	**Hex**	**Binär**
GOP-Header	000001B8	00000000 00000000 00000001 10111000

Tabelle 2.3.: GOP-Header-Startcode

Der in Tabelle 2.3 gezeigte *GOP-Header-Startcode* identifiziert den *GOP-Header*.
Dem Startcode folgt der Timecode, welcher im Format HH:MM:SS:FR vorliegt. Diese absolute Zeitinformation im Stream spielt jedoch für den Dekodierprozess keine Rolle. Noch dazu ist es nicht verbindlich, die Möglichkeit, Timecodeinformationen in den Stream zu schreiben, zu nutzen.
Die folgenden beiden Flags, *Closed GOP* und *Broken Link* geben Auskunft darüber, ob der aktuelle *GOP* Informationen vom vorangegangenen *GOP* benötigt, und ob ihm diese Informationen in Form von vorangegangenen Referenzbildern vorliegen. Mehr dazu im Kapitel 2.1.3.

2.1.3. Das *Picture* und die *Picture-Header*

Für die Pictures und Picture-Header gelten folgende Vereinbarungen[1]:

- Jede Portion von Bilddaten wird eingeleitet vom *Picture Header*.
- Ein Bild kann entweder ein I-Bild, ein P-Bild oder ein B-Bild sein. Ausschließlich in der Spezifikation ISO/IEC 11172-2[2] für MPEG1-kodierte Videodatenströme gibt es darüber hinaus auch den Bild-Typ D-Bild (intra-coded pictures with only DC coefficients), welcher in der MPEG2-Spezifikation jedoch nicht unterstützt wird. Eine Sequenz beginnt immer mit einem I-Bild.

  I-Bild (intra-coded picture): Ein I-Bild wird wie beim Einzelbild ohne Informationen von anderen Bildern verarbeitet. Daher sind sie für den Dekoder als Einstiegspunkt in den Datenstrom geeignet.

  P-Bild (predictive-coded picture): P-Bilder sind prädiktiv kodierte Bilder. Zur Dekodierung werden vorangegangene I- oder P-Bilder des selben *GOP´s* herangezogen. Dies geschieht zum Zweck der Verminderung der zeitlichen Redundanz.

  B-Bild (bidirectionally predictive-coded picture): B-Bilder werden aus dem selben Grund wie die P-Bilder interkodiert. Diese beziehen sich jedoch gleichermaßen auf vorangegangene sowie nachfolgende I- oder P-Bilder, niemals jedoch auf B-Bilder. Den Prozess nennt man „bidirektionale Prädiktion“.

  Durch die Prädiktion von Bildinhalten ist es möglich, die Bitrate des Videostromes erheblich zu verringern. Die Abbildung 2.2 zeigt den Aufbau eines *GOP´s* bestehend aus den verschiedenen Bildtypen und die untereinander herrschenden Beziehungen.

- Aufgrund der bidirektionalen Prädiktion müssen zum Zeitpunkt der Dekodierung eines B-Bildes alle Referenzbilder vorliegen, Referenzbilder müssen folglich im Vorfeld übertragen werden. Es ist also eine Variation der Speicherreihenfolge zur Abspielreihenfolge notwendig.
- Bilder werden zu *GOP´s* zusammengefasst. Hier gibt es die Typen *Offener GOP* und *Geschlossener GOP*.

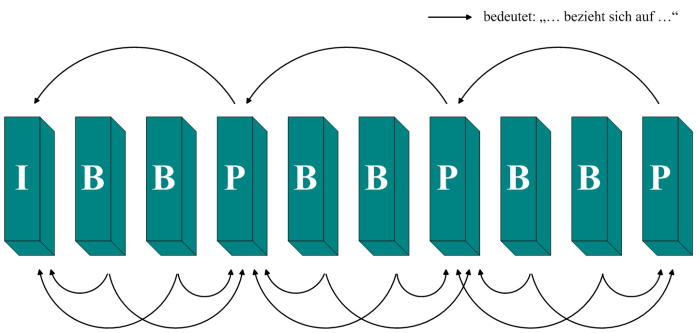

Abbildung 2.2.: Bildtypen und Beziehungen im MPEG-Videodatenstrom

Geschlossener GOP: Hier werden keine Informationen zwischen anderen *GOP´s* ausgetauscht. Dies erfordert, dass der *GOP* in der Abspielreihenfolge entweder mit einem I- oder einem P-Bild abgeschlossen werden muss. Der nächste *GOP* wird beginnen mit einem I-Bild.

Offener GOP: *Offene GOP´s* beginnen bei Betrachtung der Abspielreihenfolge mit B-Bildern, welche sich auf das letzte I- oder P-Bild aus dem letzten *GOP* und das I-Bild aus dem aktuellen *GOP* beziehen. Für die Speicherreihenfolge bedeutet dies, dass ein *Offener GOP* mit dem I-Bild beginnt, gefolgt von den *GOP*-übergreifenden B-Bildern. Eine Sequenz kann folglich nie mit einem *Offenen GOP* beginnen.

Wie bereits erwähnt befinden sich im *GOP-Header* die Parameter *Closed GOP* und *Broken Link*. Nach dem Einlesen des Parameters *Closed GOP* ist der Dekoder in der Lage, die unmittelbar dem I-Bild folgenden B-Bilder, wenn vorhanden, korrekt zu behandeln. Besitzt der Parameter *Closed GOP* den Wert Null, wird angezeigt, dass diese B-Bilder sich auf das erste I-Bild des aktuellen *GOP´s* sowie das letzte P-Bild aus dem letzten *GOP* beziehen.
Der Parameter *Broken Link* zeigt an, ob den dem I-Bild folgenden B-Bildern alle Referenzbilder vorliegen, und somit dekodiert werden können. Ist dieses Flag Eins, erhält der Dekoder die Information, dass

diese Bilder nicht dekodiert werden können und werden somit übersprungen.

Abbildung 2.3 verdeutlicht zwei Beispiel-*GOP*-Strukturen, dargestellt in der Speicherreihenfolge.

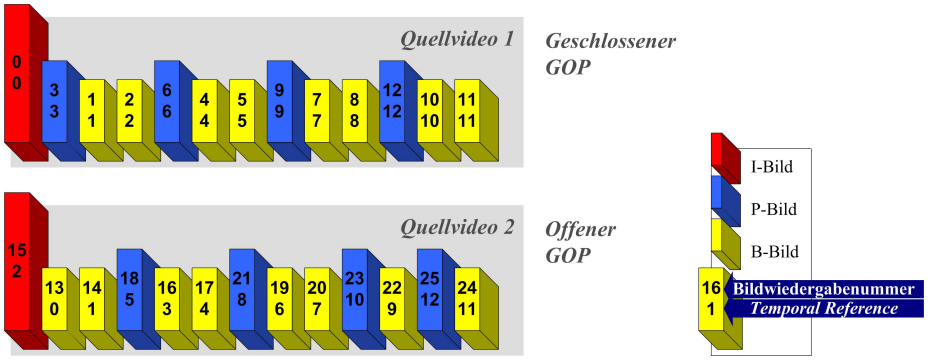

Abbildung 2.3.: *GOP*-Strukturen

- Bilder liegen im progressiven oder im Halbbildformat vor. Halbbilder können als *Frame-* oder *Field-Pictures* vorliegen.

  Progressive Frame: Bild ist ein progressives Einzelbild. Pro Bild liegt ein Datenpaket vor, es ist nur ein *Picture-Header* vorhanden.

  Frame Pictures: Bild besteht aus zwei Halbbildern. Die Daten liegen trotzdem als ein Datenpaket mit einem *Picture-Header* vor.

  Field Pictures: Bild besteht aus zwei Halbbildern. Es liegen zwei Portionen von Daten jeweils eingeleitet von einem *Picture-Header* vor. Hier muss jedes Halbbild einzeln verarbeitet werden.

Picture-Header werden eingeleitet durch den *Picture-Header-Startcode* abgebildet in Tabelle 2.4.

Startcode	Hex	Binär
Pic.-Head.-Startc.	00000100	00000000 00000000 00000001 00000000

Tabelle 2.4.: Picture-Header-Startcode

Dem *Picture-Header-Startcode* folgen folgend aufgeführte Parameter.

- Der Parameter *Temporal Reference* gibt die Position eines jeden Bildes innerhalb des GOP´s an. Hat das erste I-Bild des *GOP´s* eine *Temporal Reference* von Null, kann man davon ausgehen, dass es sich um einen *Geschlossenen GOP* handelt. Bei einer *Temporal Reference* von z. B. Zwei sollten nach dem I-Bild zwei B-Bilder mit den Werten Null und Eins folgen - dann handelt es sich um einen *Offenen GOP*.
- *Picture Coding Type* identifiziert den Bildtyp.

Es folgen weitere für diese Arbeit jedoch irrelevante Parameter.

Innerhalb eines Bildes, in der Regel nach dem *Picture-Header*, folgt die *Picture-Coding-Extension* angeführt durch den *Extension-Start-Code*. Die wichtigsten von ihm bereitgestellten Parameter sind folgend aufgeführt.

- *Picture-Structur* gibt Auskunft, ob es sich bei dem Bild um ein framebasiertes Bild, und wenn nicht, um ein fieldbasiertes oberes oder unteres Halbbild handelt.
- Das Flag *Top-Field-First* zeigt an, ob bei *Framepictures* das obere oder das untere Bild zuerst ausgegeben werden soll.

Es folgen weitere für diese Arbeit jedoch irrelevante Parameter.

2.1.4. Gültige Parameterkombinationen zwischen Quellvideomaterial

MPEG-Material wird unter Verwendung vieler verschiedener Parameter kodiert, die für diese Arbeit relevanten Parameter wurden soeben vorgestellt. Nun kann jedoch gemäß der EDL MPEG2-Material unterschiedlichster Parameter als Quellvideo zum Schnitt bereit stehen.
Die zu klärende Frage ist, in welchen Parametern MPEG-Material übereinstimmen muss, damit die Gültigkeit des Ausgangsvideos gewahrt bleibt. Nach diesen Parametern sind die Quellvideos zu untersuchen, gegebenenfalls ist der Kopierprozess zu untersagen.

Die MPEG2-Spezifikation [1] besagt, dass innerhalb einer Videosequenz alle *Sequence-Header, Sequence-Extensions* und *Sequence-Display-Extensions* die gleichen Werte zu besitzen haben.
Folgend sind die Parameter innerhalb des *Sequence-Headers*, der *Sequence-Extension* und der *Sequence-Display-Extension* aufgeführt, bei welchen die Kombinationen abweichender Werte nach der MPEG2-Spezifikation [1] verboten ist.

Sequence Header:

Horizontal & Vertical Size
Aspect Ratio Information, Framerate Code, Bitrate
VBV-Buffersize

Sequence Extension:

Low Delay
Profile & Level
Progressive Sequence
Chroma Format

Sequence Display Extension:

Color Primaries, Transfer Characteristics, Matrix Coefficients
Video Format
Display Horizontal & Vertical Size

All diese Parameter sind auf ihre Werte zu überprüfen, gegebenenfalls ist der Zusammenschnitt von Quellvideos mit abweichenden Werten dieser Parameter zu untersagen.

Mit diesem Wissen über den Aufbau von MPEG2-Elementarströmen und den dazugehörigen Parametern lassen sich nun theoretische Methoden zum Schnitt dieses Materials erarbeiten.

2.2. Technische Herausforderung beim Schnitt von MPEG-Material

Es werden theoretisch die Möglichkeiten betrachtet, MPEG2-Material möglichst effizient zu schneiden.
Im Folgenden wird häufig der Begriff „Bildwiedergabenummer“ oder „Bildnummer“ verwendet, der den Wiedergabezeitpunkt des entsprechenden Bildes meint. Weiter werden die Begriffe „Inpoint“ und „Outpoint“ verwendet. Die Inpointnummer bezeichnet das erste zu kopierende Bild einer Szene in den Ausgangsdatenstrom. Entsprechend meint die Outpointbildnummer das Bild, das als letztes Bild dieser Szene in das Ausgangsvideo zu kopieren ist. In folgend dargestellten Struktogrammen wird teilweise die Anweisung „Stelle das nächste Bild bereit“ verwendet. Die Bereitstellung der auf diese Weise angeforderten Bilder erfolgt in der Speicherreihenfolge.

2.2.1. Schnitt ohne Neukodierung

Den ressourcen- und qualitätstechnisch günstigsten Fall stellt der Schnitt ohne Neukodierung dar. Daher wird, wenn möglich, diese Schnittart anzuwenden sein. Abhängig ist die Schnittart von der Kombination zwischen In- und Outpoint.

Möglicher Inpoint: Es kommt ausschließlich das I-Bild als Inpoint in Frage, wenn nicht neu kodiert werden soll. Der Grund dafür ist, dass sämtliche weiteren Bildtypen jeweils das I-Bild als Referenzbild heranziehen.

Möglicher Outpoint: Hier kommt sowohl das I- als auch das P-Bild in Frage, da beide Bildtypen im Gegensatz zum B-Bild keine Referenzbilder besitzen, die in der Abspielreihenfolge betrachtet später als sie selbst dargestellt werden.

In allen weiteren Fällen ist eine Neukodierung der Schnittumgebung unumgänglich.
Für Schnittumgebungen ohne Neukodierungsbedarf genügt die in Abbildung 2.4 gezeigte Logik.

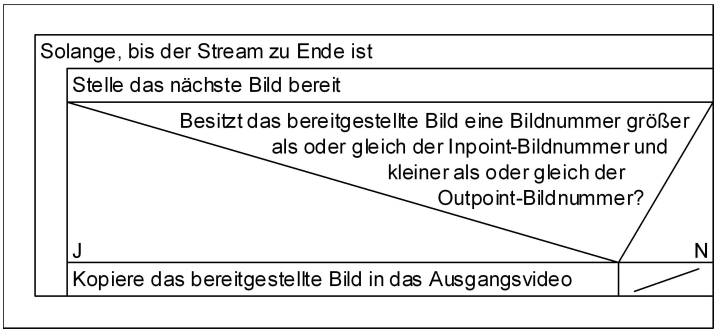

Abbildung 2.4.: Logik für Schnittumgebungen ohne Neukodierungsbedarf

Abbildung 2.5 zeigt Kopiervorgänge aus zwei Quellvideos in ein Ausgangsvideo. Der zu kopierende *GOP* aus Quellvideo Eins besitzt die *Geschlossene GOP*-Struktur, Quellvideo Zwei besitzt die *Offene GOP*-Struktur.

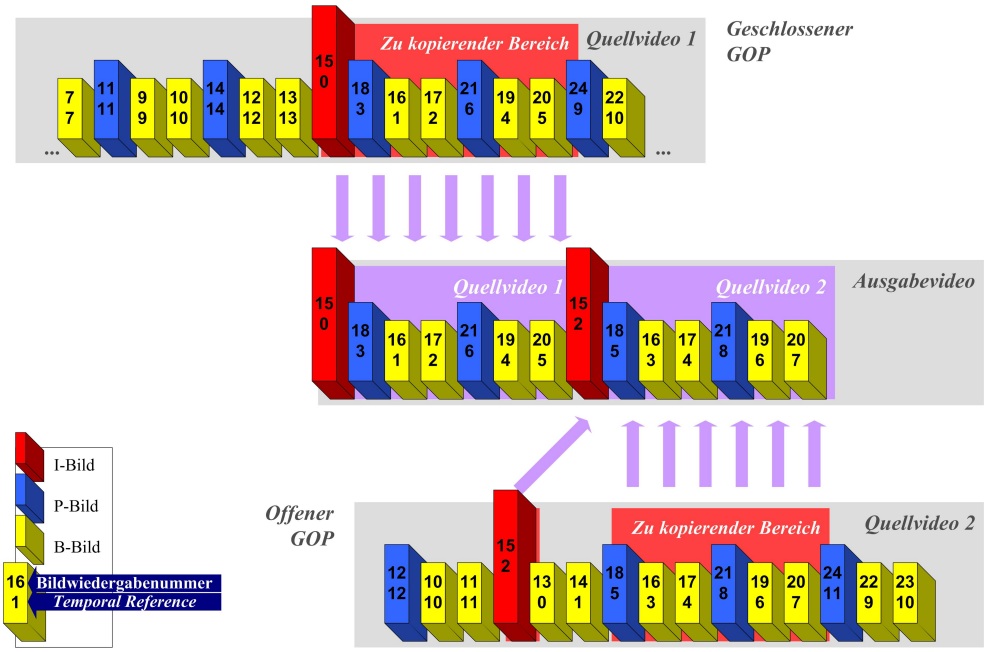

Abbildung 2.5.: Schnitt ohne Neukodierungsbedarf

Bei Betrachtung des Ausgabevideos ist deutlich das entstandene Problem der inkorrekten *Temporal References* zu sehen – der Dekoder wird dieses zusammengeschnittene Video nicht korrekt dekodieren können. Der Grund liegt in der nicht möglichen Rücksortierung der Bilder, da bei Betrachtung der Abspielreihenfolge die *Temporal Reference*-Angaben der Bilder einen

Sprung enthalten.
Desweiteren stimmt nun, wenn davon auszugehen ist, dass die *GOP-Header* vor den I-Bildern ebenfalls 1:1 mitkopiert worden sind, der *GOP-Header* der zweiten Szene in seinem Parameter *Closed GOP* nicht mehr, da es sich zu diesem Zeitpunkt im Gegensatz zum vorherigen um einen *Geschlossenen GOP* handelt. Es wird folglich eine Methode notwendig sein, welche während des Kopiervorganges bei Bedarf den Wert der *Temporal Reference* im *Picture-Header* sowie des *Closed GOP´s* im *GOP-Header* korrigiert. Abgesehen von diesen Problemen ist der Schnitt jedoch gelungen.

2.2.2. Schnitt mit Neukodierung

Für Schnittumgebungen mit Neukodierungsbedarf könnte die Logik dargestellt in Abbildung 2.6 gelten.

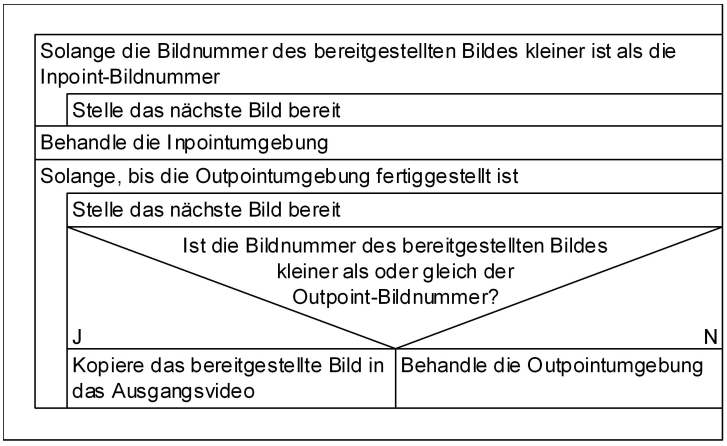

Abbildung 2.6.: Logik für Schnittumgebungen mit Neukodierungsbedarf

Aufgrund der variierten Speicherreihenfolge zur Abspielreihenfolge kommt es häufig vor, dass ein Bild mit einer höheren Bildnummer gefunden wird, obwohl das Inpoint-Bild noch nicht übertragen wurde (vgl. Abbildung 2.7). Diese Tatsache macht erforderlich, das Kopieren einzelner Bilder nicht erst zu beginnen, wenn das Inpoint-Bild vorliegt. Stattdessen ist für jedes Bild eine Entscheidung zu tätigen, ob die Bildnummer des Bildes größer als oder gleich der Inpoint-Bildnummer ist.

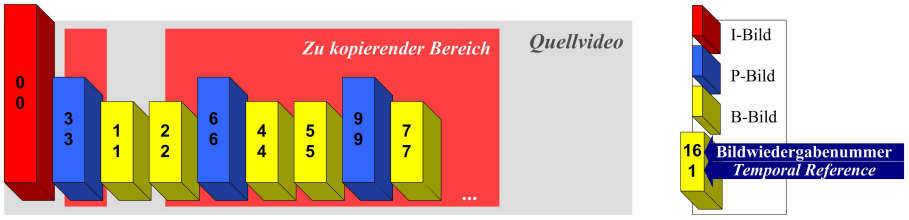

Abbildung 2.7.: Inpointumgebung mit Inpoint = 2

Eine Methode zur korrekten Behandlung des Inpoints und der Inpoint-Umgebung unter der Beachtung der Speicherreihenfolge im Vergleich zur Abspielreihenfolge sollte die in Abbildung 2.8 dargestellte Logik besitzen. Diese Methode wird aufgerufen, sobald die Bildnummer des Bildes größer als oder gleich der Inpoint-Bildnummer ist.
Weiter ist in der Abbildung 2.8 der Punkt „Sortiere und enkodiere den Enkodierbuffer“ zu sehen. Dazu ist zu sagen, dass eine Folge von Bildern nicht Bild für Bild neukodiert werden kann, da dies eine I-Frame-Only-Sequenz zur Folge hätte. Es sind also alle neu zu kodierenden Bilder in einem Speicher zwischen zu speichern. Sind alle Daten komplett, kann dieser enkodiert werden. Zuvor müssen die Bilder in diesem Speicher jedoch noch umsortiert werden, da die Bilder in der Speicherreihenfolge anstatt in der Wiedergabereihenfolge vorliegen (vgl. Kapitel 3.4.4).

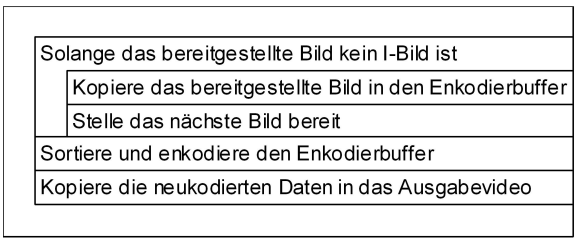

Abbildung 2.8.: Logik zur geeigneten Behandlung der Inpointumgebung

Diese Methode erfüllt fast die gewünschte Aufgabenstellung. Jedes Bild wird kopiert bis zum nächsten I-Bild, anschließend werden die Bilder für den Enkoder in die richtige Reihenfolge gebracht und enkodiert. Diese neukodierten Daten werden in das Ausgabevideo kopiert. Probleme treten jedoch auf bei Schnittstrukturen, gezeigt in Abbildung 2.9 – bei Schnitten an *Offenen GOP´s*.

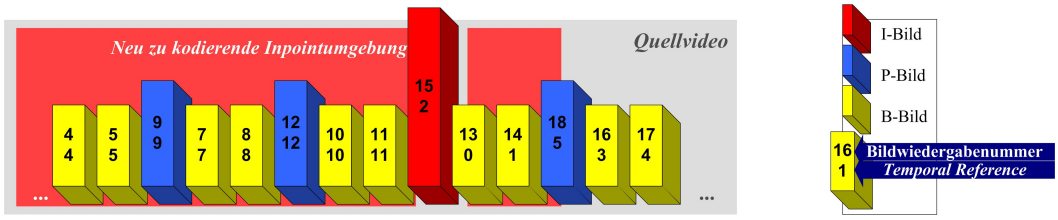

Abbildung 2.9.: Behandlung der Inpointumgebung – *Offener GOP*

Da die Rückwärtsreferenz für die Bilder 13 und 14 aufgrund der Neukodierung von Bild 12 nicht mehr gültig ist, sind diese, obwohl sie zum nächsten *GOP* gehören, ebenfalls neu zu kodieren. Dies macht auch hier erforderlich, die Bilder nicht bis zum nächsten I-Bild zu kopieren, stattdessen sind alle Bilder mit Bildnummern kleiner als die Bildnummer des nächsten I-Bildes zu kopieren. Daraus ergibt sich für die Methode zur korrekten Behandlung der Inpoint-Umgebung die in Abbildung 2.10 gezeigte Logik.

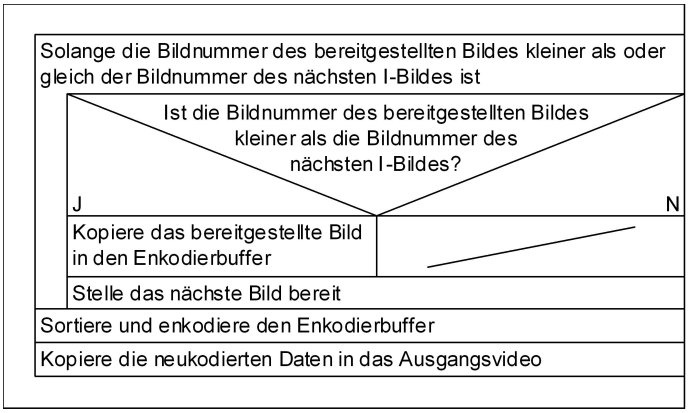

Abbildung 2.10.: Logik zur geeigneten Behandlung der Inpointumgebung

Eine Szene kann, wie bereits besprochen, auf zwei Weisen gültig abgeschlossen werden. Entweder besteht das Outpointbild aus einem I- oder P-Bild, dieses kann kopiert werden und bildet somit das letzte Bild der Szene. Bei einem B-Bild als letztes Bild ist eine Neukodierung notwendig.

Auf die Outpointumgebung kann man auf zwei Weisen treffen:

Die Bildnummer des bereitgestellten Bildes ist gleich der Outpointbildnummer: Dieses Bild ist zwangsläufig ein I- oder P-Bild. Würde es sich

um ein B-Bild handeln, würde dieses im Anschluss an seine Referenzbilder übertragen werden, dann liegt die zweite folgend beschriebene Situation vor.
Im Anschluss an dieses I- oder P-Bild besteht die Möglichkeit, dass noch B-Bilder mit kleineren Bildnummern als die Nummer des Outpointbildes vorliegen, die zu kopieren sind. Diese Situation kommt ohne eine Neukodierung der Umgebung aus.

Die Bildnummer des bereitgestellten Bildes ist größer als die Outpointbildnummer: In diesem Fall ist das Outpointbild zwangsläufig ein B-Bild, das in diesem Moment vorliegende I- oder P-Bild ist ein Referenzbild für das Outpoint-B-Bild – es ist folglich eine Neukodierung notwendig.

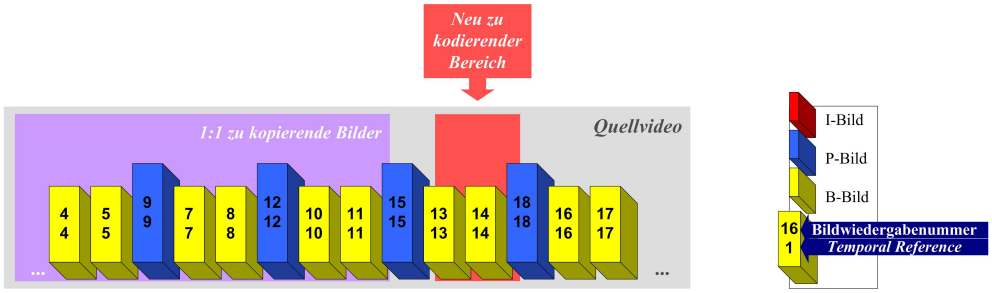

Abbildung 2.11.: Behandlung einer Outpointumgebung mit Neukodierungsbedarf

Aufgrund dieser Tatsache wird, wie in Abbildung 2.6 gezeigt, nicht in die Methode zur Behandlung der Outpointumgebung verzweigt, wenn die Bildnummer des bereitgestellten Bildes gleich der Outpoint-Bildnummer ist. Ist die Nummer des aktuellen Bildes größer als Outpoint-Bildnummer, muss die in Abbildung 2.12 dargestellte Methode zur Behandlung des Outpoints die Szene mittels Neukodierung gültig abschließen.
In Abbildung 2.11 ist eine Beispiel-Outpointumgebung aufgezeigt, welche ohne eine Neukodierung nicht gültig abzuschließen ist. Hier sind Bilder zu behandeln, welche in der Speicherreihenfolge erst nach dem eigentlichen Outpoint kommen.

Die in Abbildung 2.12 dargestellte Methode zur Behandlung des Outpoints kopiert solange die jeweils bereitgestellten Bilder in den Enkodierbuffer, bis

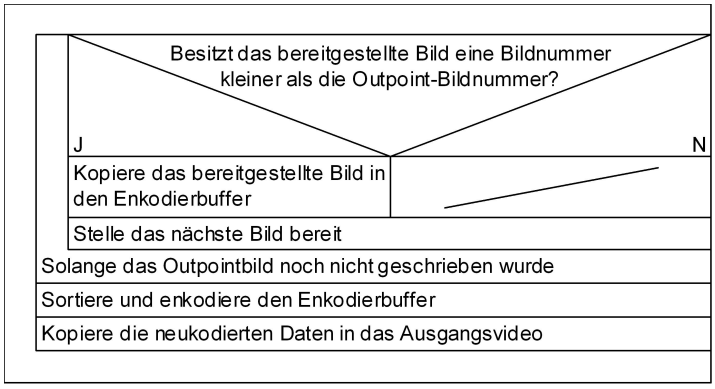

Abbildung 2.12.: Logik zur geeigneten Behandlung der Outpointumgebung

das Outpointbild selber ebenfalls kopiert wurde. Anschließend wird analog zur Methode zur Behandlung des Inpoints der Enkodierbuffer sortiert und enkodiert, im Anschluss werden die Daten in das Ausgabevideo geschrieben.

Zusammenfassung

Dieses Kapitel gab Einblicke in den Aufbau und die Struktur von MPEG2-konformen Video-Elemetarströmen. Desweiteren wurden die technischen Herausforderungen beim Schnitt solcher Strukturen theoretisch beleuchtet.

3. Material und Methoden

Dieses Kapitel beschäftigt sich mit der Methodik der Handhabung der MPEG2-Daten in der C++-Umgebung, den Intel UMC-Klassen. Anschließend wird die gewählte Objektstruktur erläutert. Die einzelnen Klassen nebst ihren Funktionalitäten werden vorgestellt sowie ausgewählte Arbeitsabläufe innerhalb des Programms kommentiert. Das Kapitel schließt ab mit der Diskussion über geeignete Methoden zum Test der entwickelten Software.

3.1. Verwendung der Intel IPP-UMC-Klassen

Um die Videodaten aus dem Datenstrom zu lesen und sie zu de- und enkodieren werden spezielle Klassen benötigt. Im Rahmen dieser Diplomarbeit werden die UMC-Klassen[1] aus der Intel-IPP-Bibliothek[2] der Version 5.1 verwendet. Abbildung 3.1 zeigt eine Beispiel-Datenstromstruktur für MPEG2-Videomaterial unter Verwendung der UMC-Klassen.

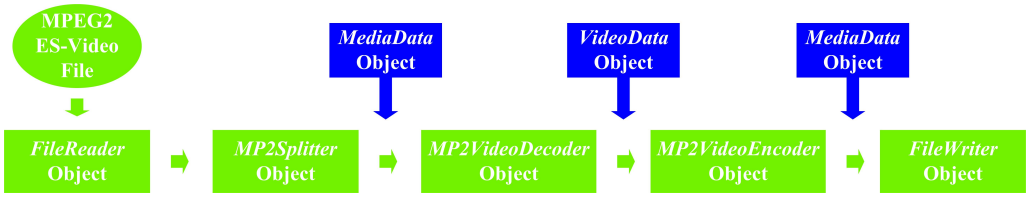

Abbildung 3.1.: Arbeitsweise der Intel UMC-Klassen[3]

Folgende UMC-Objekte sind zum Behandeln eines MPEG2-Datenstroms notwendig:

[1] UMC = Unified Media Classes
[2] IPP = Integrated Performance Primitives

MediaDataEx: Objekt, welches komprimierte Videodaten enthalten kann.

VideoData: Objekt, welches unkomprimierte Videodaten enthalten kann.

DataReader: Objekt zum Zugreifen auf den Datenstrom. Es stellt eine bestimmte Portion von Daten aus dem Datenstrom zur Verfügung

MPEG2Splitter: Objekt, welches aus den Daten vom DataReader die Daten für genau ein Bild zur Verfügung stellt. Dazu wird ein MediaData-Objekt benötigt.

MPEG2VideoDecoder: Objekt, welchem die Daten von einem komprimierten Bild in Form eines MediaData-Objektes übergeben werden. Man erhält das dekodierte Bild in Form eines VideoData-Objektes.

MPEG2Encoder: Objekt, welchem die Daten von unkomprimierten Bildern in Form von VideoData-Objekten übergeben werden. Man erhält die komprimierten Bilddaten in Form von MediaData-Objekten zurück.

DataWriter: Objekt, welchem Daten zum Schreiben in die Ausgabedatei übergeben werden können.

SplitterInfo: Objekt, welches Informationen von dem MPEG2-Splitter-Objekt bereitstellt.

MPEG2EncoderParams: Objekt, welches die Enkodierparameter für das MPEG2-Enkoderobjekt bereitstellt.

Die Zugriffe auf diese Klassen innerhalb der C++-Umgebung geschehen nach dem immer annähernd gleichen Prinzip[3]. Im folgenden Beispiel wird das MPEG2Splitter-Objekt angewiesen, die nächsten Videodaten in einem MediaData-Objekt bereitzustellen. Der Rückgabewert lässt erkennen, ob die Aktion erfolgreich war.

```
umcRes = m_CMpeg2Spl.GetNextVideoData(&m_CMediaData);
if (umcRes != UMC::UMC_OK)
  return umcRes;
```

Videodatentypen in der UMC-Umgebung Die verwendeten Videodaten innerhalb der UMC-Klassen sind eingeteilt in die komprimierten und die unkomprimierten Videodaten. Objekte vom Typ MediaData sind in der Lage, komprimierte Videodaten zu speichern, wohingegen Objekte vom Typ VideoData in der Lage sind, unkomprimierte Videodaten zu speichern. Ein MPEG2-Video zum Beispiel stellt die Videodaten in komprimierter Weise zur Verfügung, ein Enkoder beispielsweise erwartet als Eingabe Videodaten in unkomprimierter Form.
Abbildung 3.1 verdeutlicht die Zusammenhänge bezüglich der Übergabe der unterschiedlichen Videodaten-Objekttypen.

Enkodereigenschaften und Handhabung Innerhalb der UMC-Klassen wird, bis auf die Neukodierung, die Einzelbildverarbeitung angewendet. Würde man jedes Bild einzeln enkodieren, hätte es als Ausgangsdatenstrom eine I-Frame-Only-Sequenz zur Folge.
Dagegen stellt der Enkoder die Möglichkeit zur Angabe einer Anzahl der insgesamt zu enkodierenden Bilder. Mit dieser Angabe ist der Enkoder in der Lage, die einzelnen unkomprimierten Bilder sinnvoll in eine MPEG2-Bildtypstruktur aufzuteilen.
Desweiteren ist zu beachten, dass der Enkoder aufgrund der Prädiktion der Bilder und der Sortierung in die Speicherreihenfolge das eingespeiste, unkomprimierte Bild nicht unmittelbar in komprimierter Form wieder heraus gibt. Jedes Bild kann erst in den Ausgangsdatenstrom kopiert werden, wenn all seine Referenzbilder zuvor in den Ausgangsstream übertragen worden sind.

Die Handhabung von Halbbildmaterial Wie in Kapitel 2.1 bereits angedeutet, kann MPEG-Material in verschiedenen Bildformaten vorliegen: Als Progressiv- oder Halbbildmaterial. Bei Halbbildmaterial wird unterschieden zwischen framebasierten und fieldbasierten Bildern. Bei framebasierten Bildern liegen zwar Halbbilder vor, jedoch sind die Daten beider Halbbilder zu einem Datenpaket zusammengefasst, sodass sich für die Verarbeitung eines solchen Videos innerhalb der UMC-Klassen nichts ändert. Framebasiertes Halbbildmaterial lässt sich mittels der UMC-Klassen wie progressives Material verarbeiten.

Anders ist es bei fieldbasiertem Material. Hier liegen die einzelnen Halbbilder in einzelnen Datenpaketen mit jeweils einem vorangestellten *Picture-Header* vor. Zu einem Bild, bzw. einer Bildwiedergabenummer gehören also zwei Halbbilder. Folglich ist in diesem Fall statt einem MediaData-Objekt ein Feld von zwei MediaData-Objekten zu verwenden, um jederzeit die Daten für genau ein Bild bereitstellen zu können.
Möchte man nun diese beiden Halbbilder dekodieren, müssen beide Halbbilder hintereinander dem MPEG2-Dekoder übergeben werden. Nach der Verarbeitung des ersten Halbbildes lässt schon die Reaktion des Dekoders erkennen, ob fieldbasiertes Material verarbeitet wird. Der Dekoder gibt anstatt der dekodierten Daten die Meldung heraus, dass weitere Daten zu verarbeiten sind. In dem Fall ist im nächsten Schritt das zweite Halbbild zu verarbeiten.

3.2. Objektstruktur und Beziehungen

3.2.1. Objektstruktur

In Abbildung 3.2 ist die Objektstruktur der entwickelten Software dargestellt. Zu sehen ist, in welcher Beziehung zueinander die Ein- und Ausgabevideofiles, die EDL und die verwendeten Objekte innerhalb des Programms stehen.

Der *Manager* übernimmt die Kontrolle des gesamten Prozesses. In seinen Prozessschleifen weist er die entsprechenden Objekte zur Verarbeitung des Schnittauftrages an.

Das *CSource*-Objekt repräsentiert ein Quellvideo mit allen notwendigen Eigenschaften und Funktionalitäten. Für jedes Quellvideo wird zu Beginn im *Manager* je ein Objekt vom Typ *CSource* erzeugt, sodass parallel alle Quellvideos in Form dieser Objekte zur Verfügung stehen.

Die Klasse *CScene* wird für jede zu verarbeitende Szene erzeugt und ist auch nur solange gültig, bis die Szene bis zum Ende verarbeitet wurde. Sie vereint alle szenenintern benötigten Funktionalitäten und Eigenschaften.

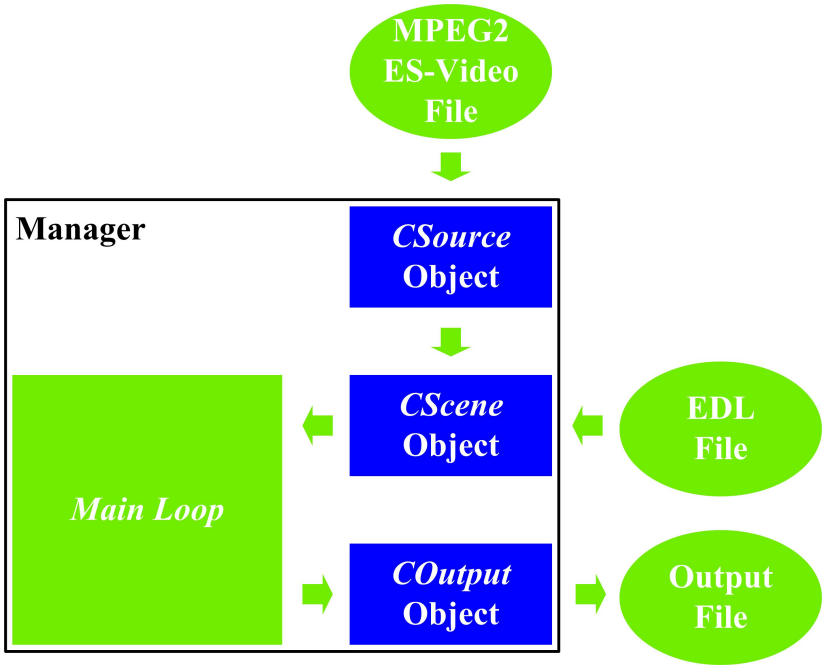

Abbildung 3.2.: Allgemeine Objektstruktur

Sie versteht sich ebenfalls als Schnittstelle zwischen dem Manager des Hauptprogramms und jeweils einem Objekt der Klasse *CSource*, und stellt somit das für diese Szene erforderliche Quellvideo bereit.

Ein Objekt der Klasse *COutput* repräsentiert das Ausgabevideo. Es vereint Funktionalitäten wie das Schreiben der Ausgangsdaten, die Neukodierung, und das Schreiben des Timecodes. Es wird insgesamt ein Objekt dieser Klasse vom *Manager* erzeugt.

3.2.2. Steuer- und Videodatenaustausch

Abbildung 3.3 demonstriert den Daten- und Informationsaustausch innerhalb des Programms. Zweckmäßigerweise wird mit Einzelbildern gearbeitet, da für jedes Bild verschiedene Fallunterscheidungen notwendig sind. Desweiteren unterstützen die UMC-Klassen diese Arbeitsweise.
Das jeweils nächste Bild wird vom Objekt der Klasse *CScene* angefordert und entsprechend der Bildnummer zum Objekt vom Typ *COutput* weitergeleitet. Dort kann die Bildinformation auf zwei Wegen in das Ausgangsvideo

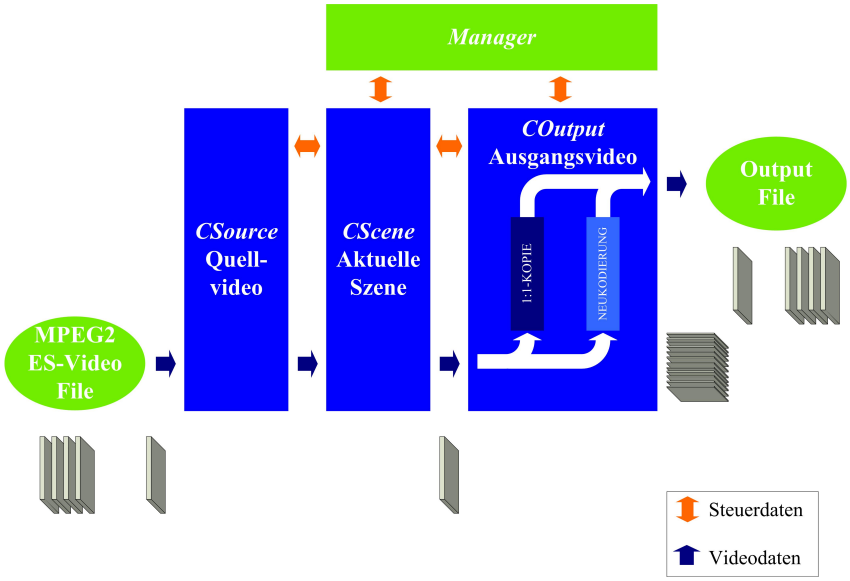

Abbildung 3.3.: Daten- und Informationsaustausch

gelangen. Entweder, wenn kein Neukodierungsbedarf besteht, als 1:1-Kopie, oder, wenn Neukodierungsbedarf besteht, über den Enkodierbuffer, welcher alle neuzukodierenden Bilder zwischenspeichert. Ist diese Portion der neu zu kodierenden Bilder komplett, werden diese Bilder enkodiert (vgl. Kapitel 3.4.4). Anschließend werden sie in das Ausgangsvideo kopiert.

3.2.3. Videodatentypen und Datenfluss

Innerhalb des Programms werden zwischen den Objekten und dem *Manager* verschiedene Videodatentypen ausgetauscht. Die Unterteilung in diese Videodatentypen werden in den UMC-Klassen festgelegt (vgl. 3.1). Abbildung 3.4 visualisiert den Datenfluss des gesamten Programms unter Verwendung der zwei verschiedenen Datentypen.

Aus Zeit- und Qualitätsgründen wird, wenn möglich, auf eine Neukodierung und die dadurch erforderliche Dekodierung verzichtet. Wann immer Bilder 1:1 kopiert werden können, entspricht der Datenfluss dem in Abbildung 3.4 gezeigten oberen horizontalen Weg. Hierbei ist mit einer enormen Zeitersparnis keine Transkodierung der komprimierten Videodaten in die unkomprimierten und zurück erforderlich.

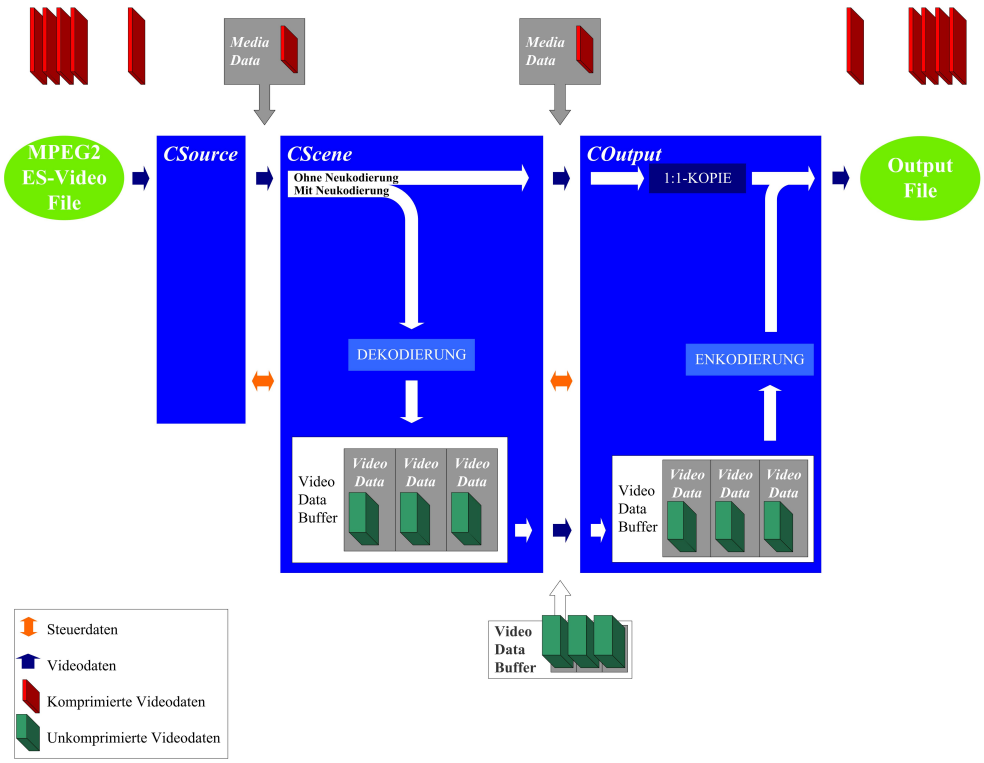

Abbildung 3.4.: Datenfluss der komprimierten und unkomprimierten Videodaten

Im Falle einer notwendigen Neukodierung wird der untere Prozessweg in der Abbildung beschritten. Der untere Prozessweg arbeitet im Gegensatz zum oberen nicht mit Einzelbildern. Im Vorfeld ist die neuzukodierende Portion von Bildern in unkomprimierter Form in einem geeigneten Buffer zwischenzuspeichern, diese ist dem *COutput*-Objekt zum Neukodieren der Videodaten zu übergeben. Die enkodierten Daten, mittlerweile wieder in Form eines MediaData-Objektes, werden im Anschluss an die Neukodierung in das Ausgangsvideo geschrieben.

Bei Betrachtung eines nach diesem Ablauf geschnittenen Videos fällt auf, dass die Bilder in den neukodierten Bereichen falsch sortiert sind. Der Grund liegt in der variierten Speicherreihenfolge zur Abspielreihenfolge. Bei der 1:1-Kopie stellt dies kein Problem dar, da das Ausgabevideo bei linearer Kopie der Bilder dem Eingangsvideo entspricht – die Rücksortierung übernimmt der Player, der dieses Videofile abspielt. Bei der Dekodierung hinge-

gen werden die Bilder vom UMC-Dekoder nicht zurücksortiert, sodass die unkomprimierten Bilder im Enkodier-Buffer unsortiert vorliegen. Der Enkoder enkodiert diese unsortierte Portion von Bildern.
Dieses Problem lässt sich durch eine Rücksortierung der Bilder im Enkodier-Buffer vor der Enkodierung lösen.

3.2.4. Die Klassen und ihre Funktionalitäten

Abbildung 3.5 stellt die entwickelten Klassen nebst ihren wichtigsten Methoden und Eigenschaften vor. Zusätzlich werden die verwendeten Objekte dargestellt.

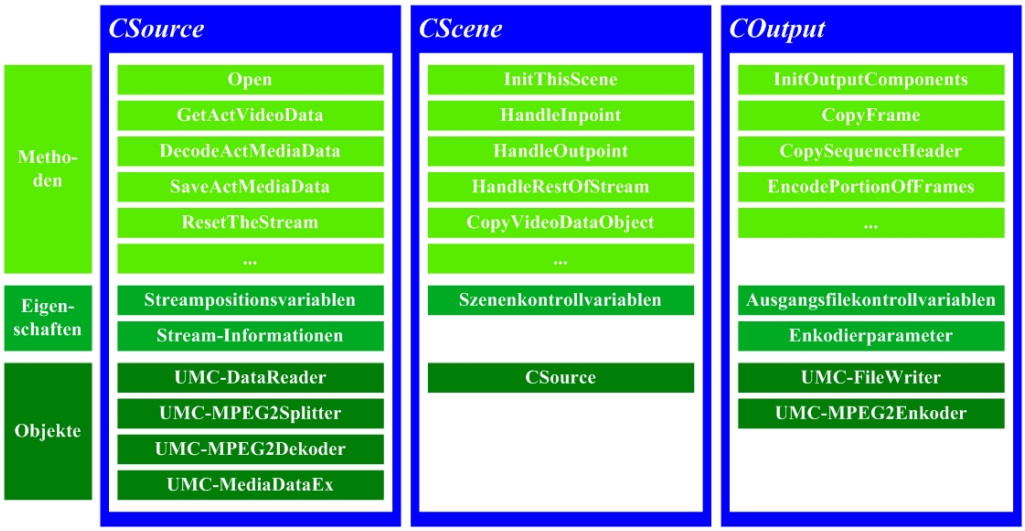

Abbildung 3.5.: Die Klassen und ihre Methoden und Eigenschaften

CSource Diese Klasse repräsentiert ein Quellvideo. Die Methoden und Eigenschaften ermöglichen die Beschaffung und Dekodierung der aktuellen Videodaten, die Berechnung der absoluten Bildnummer des aktuellen Bildes im Datenstrom, das Zurücksetzen des Datenstromes sowie das Analysieren des Datenstromes auf relevante MPEG-Parameter. All dies wird ermöglicht durch die inkludierten UMC-Klassenobjekte, welche den Zugriff auf den Stream ermöglichen.

CScene Ein Objekt dieser Klasse ist verantwortlich für die Verarbeitung genau einer Szene. Sie stellt Methoden bereit zur Initiierung der Szene, zur korrekten Behandlung von Inpoint- und Outpointumgebungen und zur Steuerung des Datenflusses innerhalb des Programms. Sie fordert mittels eines *Source*-Objektes die Mediadaten an und leitet sie an dementsprechende Methoden weiter, je nach Situation in komprimierter oder unkomprimierter Form.

COutput Ein Objekt dieser Klasse sorgt mittels der Methode *InitOutputComponents* dafür, dass der Weg in das Ausgangsvideo erstellt wird. Über diesen Weg leitet sie die ihr übertragenen Videodaten in die Datei. Dies geschieht entweder 1:1 über die Methode *CopyFrame*, oder über die Methode *EncodePortionOfFrames*, welche einen Buffer von unkomprimierten Videodaten neu kodiert und anschließend in die Datei weiterleitet. Mit den Ausgangskontrollvariablen ist diese Klasse immer über den aktuellen Zustand des Ausgabevideos informiert.

3.3. Der Workflow des Managers

Wie bereits beschrieben übernimmt der *Manager* die zentrale Rolle innerhalb des Programms. Beim Start erzeugt er zu jedem Quellvideo je ein Objekt vom Typ *CSource*. Desweiteren wird ein Objekt vom Typ *COutput* erzeugt, welches das Ausgangsvideo repräsentiert.
Es folgt die Haupt-Schleife, deren Workflow Abbildung 3.6 zeigt. Im Folgenden werden die einzelnen gezeigten Unterpunkte allgemein erläutert.

- Initiierung der Szene
  Hier wird das Szenenobjekt erzeugt und die Parameter werden auf den entsprechenden EDL-Eintrag gesetzt. Desweiteren wird hier dem Szenenobjekt das entsprechende Quellobjekt vom Typ *CSource* zugewiesen, sodass im Anschluss die Verarbeitung der Szene beginnen kann.

- Suchen des Inpoints
  Das Objekt *CScene* wird vom *Manager* angewiesen, den nächsten Inpoint zu suchen. Dabei muss der Stream für den bildgenauen Schnitt

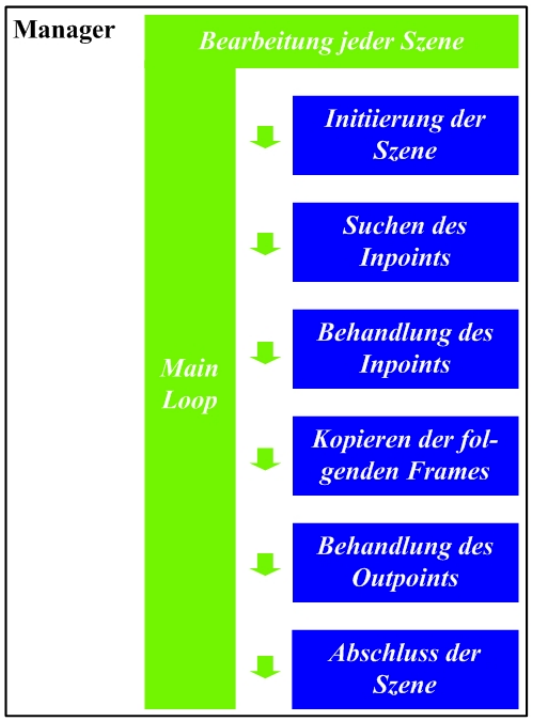

Abbildung 3.6.: Workflow des Managers

von Anfang an, bzw. von der aktuellen Position Bild für Bild durchlaufen werden. *CScene* fordert so lange das nächste Bild an, bis der Inpoint erreicht wurde. Unter speziellen Voraussetzungen können einzelne Bilder auch direkt angesprungen werden (vgl. Kapitel 3.4.1).

- Behandlung des Inpoints
  Ist der Inpoint gefunden, muss für die korrekte Behandlung der Inpointumgebung gesorgt werden. Hier wird ebenfalls in *CScene* die Situation erkannt und entsprechend der Situation reagiert. Bei Neukodierungsbedarf wird hier eine Neukodierung eingeleitet. Es ist ebenfalls möglich, dass die Inpointumgebung gleichzeitig den Outpoint mit einschließt, wenn dieser entweder in dem neuzukodierenden Bereich liegt, bzw. die Nummer des Inpointbildes gleich der Outpoint-Bildnummer ist.

- Kopieren der folgenden Bilder
  CScene behandelt das weitere Kopieren der Bilder in den Ausgangsstream. Auswerteverfahren erkennen anhand der Bildnummer und des

Bildtyps, ob der Outpointbereich erreicht ist. Ist dies der Fall wird ebenfalls erkannt, ob eine Neukodierung am Ende dieser Szene notwendig ist. Falls nicht, ist keine explizite Behandlung des Outpoints notwendig.

- Behandlung des Outpoints
  Falls Neukodierungsbedarf für die Outpointumgebung besteht, werden hier die entsprechenden Bilder neu kodiert und in den Ausgangsstream kopiert.

- Abschluss der Szene
  Die Szene wird abgeschlossen mit dem Löschen aller nicht mehr benötigten Objekte und Buffer.

3.4. Erläuterung angewendeter Methoden und Techniken

3.4.1. Der Suchmechanismus

Die Eigenschaften des MPEG-Systems und der UMC-Klassen ermöglichen unter speziellen Voraussetzungen einen direkten Zugriff auf bestimmte Bilder im Videodatenstrom. Mögliche Bilder für den Einstieg sind I-Bilder mit vorangestellten *Sequence-Headern*. Bei nicht wiederholten *Sequence-Headern* im Datenstrom ist diese Technik folglich nicht einsetzbar.
Werden die *Sequence-Header* wiederholt, kann der MPEG2-Splitter an die Stelle des gewünschten Einstiegsbildes positioniert werden. Dazu wird allerdings die absolute Position der einzelnen Bilder im Datenstrom benötigt, diese Informationen sind zuvor zu ermitteln. Daher wird während der Verwendung der Videodaten für jede Videodatei je ein Index geschrieben, in den, sollte ein I-Bild mit vorangestelltem *Sequence-Header* vorliegen, die absolute Position dieses Bildes gespeichert wird. Liegt ein Eintrag für ein I-Bild bereits vor, wird kein neuer Eintrag erstellt – so werden während der Verwendung der Videodateien die Indizes mit den absoluten Positionen der I-Bilder ständig ergänzt. Bei weiteren Verwendungen dieser Videodateien kann dann auf diese Indizes zurückgegriffen und die Streams können ent-

sprechend positioniert werden.
Werden die *Sequence-Header* nicht wiederholt, findet trotzdem eine weitere Maßnahme zur Zeitersparnis ermöglicht durch separate Videofile-Objekte Anwendung: Die Wiederaufnahme der Suche innerhalb eines bereits teilweise benutzten Videofile-Objektes. An der Stelle, an der zuletzt auf die Videodatei zugegriffen wurde, verharrt das Videofile-Objekt, bis es zum nächsten mal aufgerufen wird. Anschließend kann ab der aktuellen Position der Stream weiter zur Suche des nächsten Inpoints durchlaufen werden. Liegt der Inpoint vor der aktuellen Stelle im Stream, muss der Stream jedoch zurückgesetzt und von neuem durchlaufen werden.
Diese Maßnahme bringt die größte Zeitersparnis, wenn sich die Szenen im Quellvideo linear von vorne nach hinten ohne Überlappung von Bildern verteilen. Den ungünstigsten Fall stellt jedoch eine Verteilung der Szenen im Quellvideo von hinten nach vorne dar, da hier zu jedem Szenenwechsel ein Zurücksetzen des Streams notwendig ist, und ein erneutes Durchlaufen viel Zeit kosten würde.

3.4.2. Die Berechnung der Bildwiedergabenummern

MPEG-Streams besitzen an keiner Stelle eine verlässliche Information über die absolute Position im Stream. Einzig der *GOP-Header* hält die Option bereit, den absoluten Timecode zu speichern, es ist jedoch nicht zwingend, diesen zu verwenden. Es muss also eine andere Möglichkeit gefunden werden, die Bildnummer aller Bilder im Datenstrom sicher zu identifizieren.
Die einzige Möglichkeit, damit dies gelingt, ist, Bild für Bild durch den Stream zu wandern und eigenhändig mitzuzählen. Dabei ist der Zählmechanismus nicht trivial, da die Bilder nicht in der Abspielreihenfolge vorliegen. Anhand des Bildtyps und der *Temporal Reference* des aktuellen Bildes, sowie der bisherigen Bilder muss es möglich sein, den absoluten Wiedergabezeitpunkt eines Bildes zu errechnen.
Diese Aufgabe übernimmt die Methode *GetActVideoData* der Klasse *CSource*. Sie fordert das jeweils nächste Bild von den UMC-Klassen an. Aus den angeforderten Daten liest sie außerdem die *Temporal Reference* (vgl. 2.1.3). Desweiteren liegt ihr vor, welchen Bildtyp das aktuelle Bild besitzt, wann das letzte Bild eine *Temporal Reference* von Null besaß und welche maxima-

le Bildwiedergabenummer bisher auftrat. Mit diesen Werten lässt sich nach dem Struktogramm in Abbildung 3.7 die absolute Bildnummer berechnen.

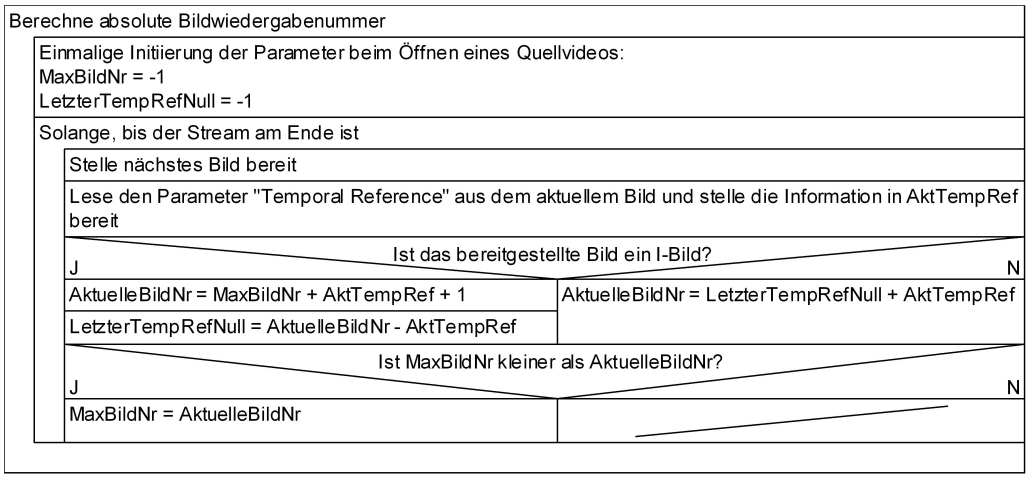

Abbildung 3.7.: Berechnung der absoluten Bildwiedergabenummer eines Bildes

Beim erstmaligen Öffnen des entsprechenden Quellvideos werden die Werte *MaxBildNr* und *LetzterTempRefNull* auf -1 gesetzt, damit der Start in die Berechnung der Bildnummern gelingt. Da jede gültige MPEG2-Sequenz mit einem I-Bild beginnt, muss beim ersten Durchgang folglich der linke Verarbeitungszweig betrachtet werden. Weiter zu sehen ist nun die Berechnung der I-Bildnummer. Da das erste Bild eine *Temporal Reference* von Null besitzt, wird als Ergebnis die Bildnummer Null erhalten. Die letzte Bildnummer, an welcher die *Temporal Reference* zu Null wurde, nämlich in diesem Moment, ist ebenfalls Null.
In den nächsten Durchgängen wird nun jeweils die Bildnummer berechnet durch das letzte Bild, welches die *Temporal Reference* von Null besaß plus der aktuellen *Temporal Reference*. Weiter wird hier die maximal beobachtete Bildnummer zwischengespeichert.
Beim Auftreten des nächsten I-Bildes berechnet sich dieses wie bereits beschrieben durch die maximal beobachtete Bildnummer plus der aktuellen *Temporal Reference* plus Eins. Die maximal beobachtete Bildnummer ist in jedem Falle das letzte Bild des letzten *GOP´s*, erhöht man diesen Wert um Eins ist also das erste Bild des aktuellen *GOP´s* berechnet.

Nun kann es sich bei diesem *GOP* jedoch auch um einen *Offenen GOP* handeln (vgl. 2.1.3), das erste I-Bild besitzt also eine *Temporal Reference* größer als Null. Diesem Verhalten entspricht die Addition der aktuellen *Temporal Reference* auf die Bildnummer mit der *Temporal Reference* von Null.
Im Anschluss wird erneut der Wert *LetzterTempRefNull* berechnet. Entsprechend der *GOP*-Eigenschaften wird auch der Wert berechnet durch die aktuelle Bildnummer minus der *Temporal Reference*. Der Wert *LetzterTemprefNull* beschreibt also nicht die Bildnummer des letzten I-Bildes, sondern des letzten Bildes mit der *Temporal Reference* von Null.
Mit dieser Methodik lässt sich sicher und einfach die absolute Bildnummer bestimmen, unter der Vorraussetzung, dass der Stream von Anfang an durchlaufen wird.

3.4.3. Notwendige Modifikationen im Stream

Bei der 1:1-Kopie von komprimierten Videodaten kann in Verbindung mit *Offenen GOP´s* der in Abbildung 3.8 beschriebene Fall auftreten. Dieser Fall ist möglich, wenn entweder der Inpoint Bildnummer 15 ist, oder der Inpoint kleiner der Bildnummer 15 ist, mit der Folge, dass die Inpointumgebung bis Bild 15 neukodiert wird. In beiden Fällen zählen die Bilder der Wiedergabenummer 13 und 14 nicht zu dem 1:1-kopierten Bereich.

Wie bereits im Kapitel 2.2 beschrieben ist in diesen Fällen zum einen eine Modifikation der *Temporal Reference* und zum anderen des *Closed GOP* notwendig. Beide Modifikationen kommen in der Methode *CopyFrame* aus der Klasse *COutput* zum Einsatz. An dieser Stelle werden die komprimierten Daten in das Ausgangsvideo geschrieben.
Die *Temporal Reference*-Modifikation geht folgendermaßen vor: Vom kopierten I-Bild wird die *Temporal Reference* ausgelesen und als Variable *TempRefOffset* gespeichert. Anschließend wird der Wert der *Temporal Reference* im I-Bild mit dem Wert Null überschrieben. Von allen weiteren Bildern bis zum nächsten I-Bild wird der Wert der Variablen *TempRefOffset* vom Wert der *Temporal Reference* des jeweiligen Bildes abgezogen. Das Ergebnis dieser Prozedur zeigt ebenfalls Abbildung 3.8.
Die *Closed GOP*-Korrektur setzt das entsprechende Flag im *GOP*-Header.

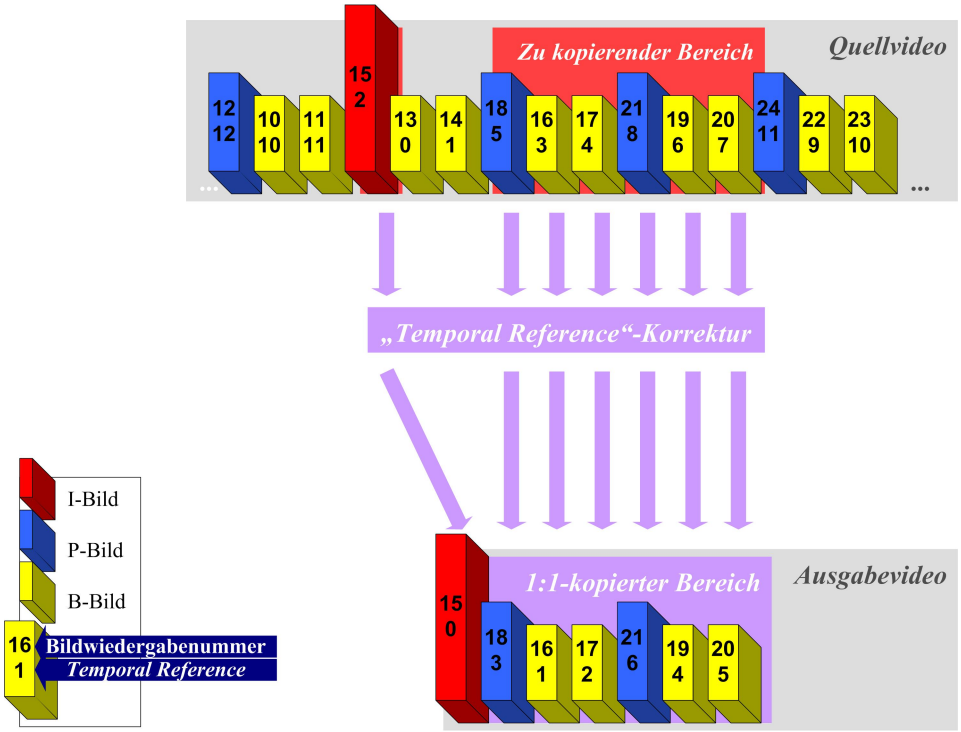

Abbildung 3.8.: Wirkungsweise der „Temporal Reference“-Modifikation

Wenn, wie im Beispiel gezeigt, ein *Offener GOP* zu einem *Geschlossenen GOP* verringert wurde, wird das Flag *Closed-GOP* gesetzt.
Als weitere Modifikation des Streams ist die Timecode-Korrektur aufzuführen, welche auf den *GOP-Header* angewendet wird. Der Timecode im *GOP-Header* ist zwar optional, jedoch soll vermieden werden, dass eventuell im Quellvideo vorhandener und in den Ausgangsstream kopierter Timecode dem in Zukunft folgenden Dekodierprozess falsche Timecodeinformationen vermittelt.
Der Timecode liegt im *GOP-Header* in dem Format HH:MM:SS:FR vor. Die absolute aktuelle Bildnummer des Ausgangsvideos, welche vom *COutput*-Objekt bereitgestellt wird, wird innerhalb der Methode *CopyFrame* aus der Klasse *COutput* in das TC-Format umgerechnet und in den *GOP-Header* geschrieben. Eventuell vorhandener Timecode wird somit überschrieben. Diese Prozedur findet ebenfalls bei neukodiertem Material Anwendung, sodass das komplette Ausgangsvideo eine gültige Timecodespur in den *GOP-Headern* bereitstellt.

3.4.4. Zweck und Verwendung der eingesetzten Buffer

Im Folgenden werden die verschiedenen eingesetzten Buffer erläutert. Abbildung 3.9 zeigt die Zusammenhänge der Buffer während der Behandlung einer Szene mit Neukodierungsbedarf. Es ist als Inpoint Bildnummer 4 und als Outpoint Bildnummer 21 gewählt.

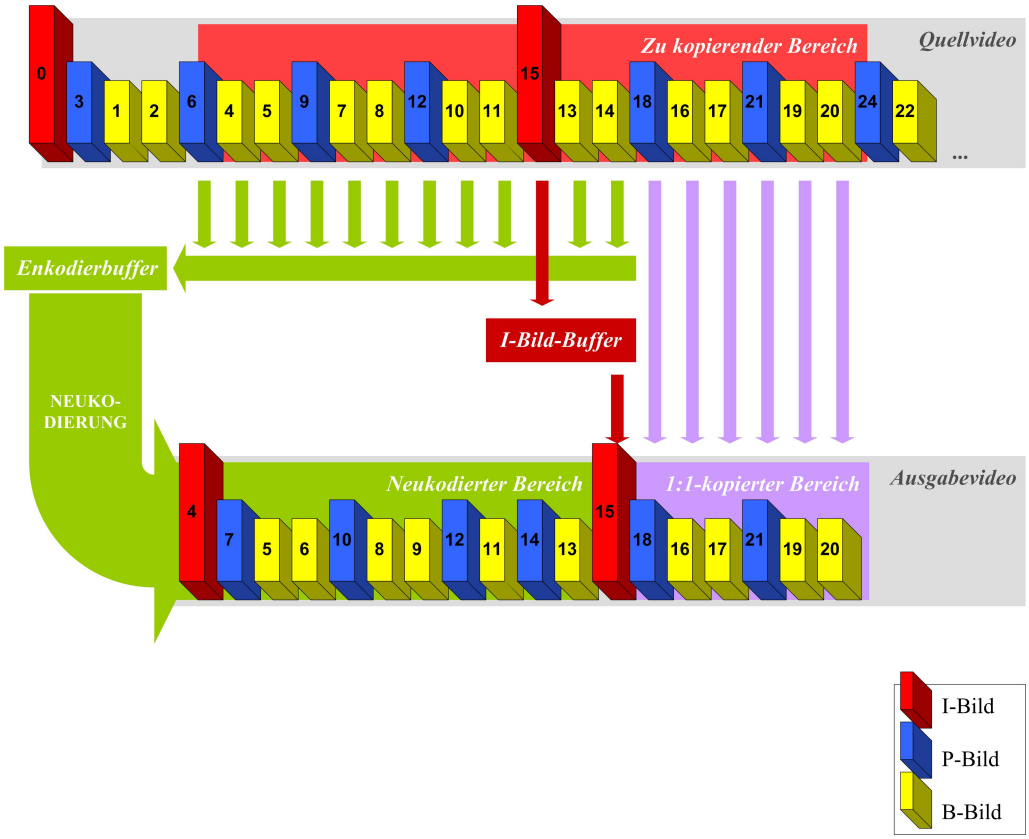

Abbildung 3.9.: Zusammenhänge der unterschiedlichen Buffer

Der I-Bild-Buffer

Wie im Kapitel 2.2 bereits beschrieben, herrscht bei *Offenen GOP´s* die Situation, dass bei Betrachtung der Speicherreihenfolge nach einem übertragenen I-Bild, welches als erstes Bild nicht mehr neukodiert zu werden braucht, bis zum nächsten I- oder P-Bild noch eine variable Anzahl von B-Bildern übertragen werden können. Abbildung 3.4.4 verdeutlicht diese Situation.
Bei Betrachtung der Abspielreihenfolge liegen diese Bilder vor dem bereits

übertragenen I-Bild. Das P-Bild ist jedoch neu kodiert worden, somit ist dieses Referenzbild für besagte B-Bilder verloren. Sie sind folglich ebenfalls neu zu kodieren, dazu werden sie ebenfalls in den Enkodierbuffer kopiert. Das bereits übertragene I-Bild ist im Anschluss an den neu kodierten Bereich zu kopieren, im Gegensatz zu vorher also hinter die beiden B-Bilder. Um dies zu erreichen wird das I-Bild im I-Bild-Buffer zwischengespeichert. Im Anschluss werden die B-Bilder in den Enkodierbuffer kopiert, dieser wird enkodiert und die neu kodierten Daten werden in das Ausgabevideo kopiert. Nun kann das zwischengespeicherte I-Bild als 1:1-Kopie in den Stream übertragen und die folgenden Bilder weiter verarbeitet werden.

Der Enkodierbuffer

Ebenfalls in Abbildung 3.4.4 zu sehen ist der Enkodierbuffer. In ihn gelangen die Bilder, die notwendigerweise neu kodiert werden müssen, wenn Referenzbilder nicht mehr vorhanden sind. Die Videodaten werden im Enkodierbuffer unkomprimiert gespeichert. Bevor die Bilder im Enkodierbuffer neukodiert werden können, ist eine Rücksortierung der Bilder notwendig. Bei Betrachtung der Abbildung 3.4.4 wird dies schnell deutlich, da die Bilder nach der Speicherreihenfolge in den Enkodierbuffer kopiert werden.
Die Rücksortierung arbeitet nach folgendem Prinzip: Soll ein Bild einer bestimmten Bildnummer in den Enkodierbuffer kopiert werden, wird kontrolliert, ob das zu kopierende Bild eine Bildnummer größer oder kleiner dem zuletzt gespeicherten Bild besitzt. Ist die Bildnummer größer, wird das Bild an das Ende des Enkodierbuffers kopiert, ist sie kleiner, wird es vor das letzte Bild im Enkodierbuffer kopiert. Nach diesem Prinzip kann jede beliebige MPEG-konforme Bildtypstruktur in die Abspielreihenfolge zurücksortiert werden.
Es folgt nun die Neukodierung der sich im Enkodierbuffer befindenden korrekt sortierten Bilder. Nach der Neukodierung wird kaum ein Bild den Bildtyp besitzen, den er vor der Dekodierung besessen hat. Dies erklärt die in Abbildung 3.4.4 zu beobachtende Abweichung des Bildtyps eines Bildes vor und nach der Neukodierung.

Der Dekodierbuffer

Sollen beispielsweise zwei einzelne B-Bilder neu kodiert werden, wird es nicht genügen, den Dekoder diese beiden Bilder dekodieren, und anschließend vom Enkoder enkodieren zu lassen. Der Grund hierfür sind die dem Dekoder nicht vorliegenden Referenzbilder, auf welche sich die B-Bilder beziehen. Folglich muss sichergestellt sein, dass, bevor Bilder zum Enkodieren dekodiert werden, der Dekoder alle benötigten Referenzbilder ebenfalls dekodiert hat.
Dies ließe sich auf folgende Weise mit mäßigem Zeitverlust realisieren: Wird während des Suchens des Inpoints eine Portion von Daten als I- oder P-Bild identifiziert, werden diese Rohdaten in einen Buffer kopiert. Nach dem ersten P-Bild nach einem I-Bild wird bis auf die letzten drei Bilder der Buffer wieder freigegeben (es stehen jetzt also ein P-Bild, ein I-Bild und ein P-Bild im Buffer). Ist jetzt eine Umgebung neu zu kodieren, wird der Dekoder zuvor veranlasst, die im Buffer gespeicherten Bilder zu dekodieren. Anschließend hat der Dekoder alle Referenzbilder der anstehenden neu zu kodierenden Umgebung dekodiert und ist somit bereit zum Dekodieren aller folgender Bilder.
Mit dieser Maßnahme ist der Dekoder jederzeit in der Lage, Bilder korrekt zu dekodieren, obwohl eine nicht unerhebliche Anzahl vorangegangener Bilder nicht dekodiert wurde. Als Zeitverlust ist zu rechnen die Zeit des Kopierens der relevanten komprimierten Videodaten im internen Speicher, sowie die Zeit zum Dekodieren. Es ist zu bemerken, dass nach diesem Prinzip kaum mehr Dekodiervorgänge als nötig vollzogen werden.
Mit der in diesem Programm nicht umgesetzten Maßnahme, anstatt des Kopierens der Daten im internen Buffer eine parallele MediaData-Objektstruktur aufzubauen, könnte noch dazu der Gesamtzeitaufwand um das Kopieren dieser Daten minimiert werden, jedoch die Art und Weise der Realisierung der UMC-Mediaklassen schließen eine derartige Arbeitsweise aus.

3.4.5. Die Handhabung von Halbbildmaterial

Kapitel 3.1 beschrieb bereits die Handhabung von Halbbildmaterial mittels der UMC-Klassen. An dieser Stelle soll die technische Umsetzung dieser

Problemlösung erläutert werden.
Generell wird innerhalb der entwickelten Software mit einem Feld von zwei MediaData-Objekten gearbeitet. Wird nun mit Progressivmaterial oder framebasiertem Halbbildmaterial gearbeitet, wird ausschließlich das MediaData-Objekt der Feldnummer Null verwendet. Sollte die Dekoderroutine feststellen, dass nach der ersten Portion der Daten noch weitere zu dem aktuellen Bild gehörige Daten folgen, handelt es sich um fieldbasiertes Halbbildmaterial. Die Weiterverarbeitung des Bildes wird nicht gestartet, sondern zuerst das zweite MediaData-Objekt mit dem zweiten Halbbild aufgefüllt. Die Portion der Daten für genau ein Bild ist komplett und kann nun verarbeitet werden.
Im Anschluss erkennen die Ausleseverfahren automatisch, wie viele Felder mit Mediadaten gefüllt sind. Ist nur das erste Feld mit Daten gefüllt, wird auch nur das erste verarbeitet.
Auf diese Weise ist die Software in der Lage, sich dynamisch an das vorhandene Videomaterial anzupassen.

3.5. Struktogramme ausgewählter Methoden

Im folgenden werden ausgewählte Methoden verschiedener Klassen dargestellt und ihre Funktion und Arbeitsweise beschrieben.

3.5.1. *FindNextInPoint*

Die Methode *FindNextInPoint*, integriert in die Klasse *CScene*, folgt dem in Abbildung 3.10 aufgezeigten Struktogramm. Mittels einer Do-While-Schleife wird der Stream positioniert. Innerhalb der Schleife wird das jeweils nächste Bild vom Quellvideo bereitgestellt und auf den Bildtyp analysiert. Handelt es sich um ein I- oder P-Bild wird es in den Dekodierbuffer (siehe Kapitel 3.4.4) kopiert. Die Schleifenbedingung überprüft, ob das bereitgestellte Bild eine Bildnummer größer als oder gleich der Inpoint-Bildnummer besitzt. Dieser Fall ist abzufragen, da es aufgrund der variierenden Speicherreihenfolge zur Abspielreihenfolge möglich ist, dass vor dem eigentlichen Inpointbild Bilder mit höherer Bildnummer übertragen werden können (siehe Kapitel 2.2.2). Besitzt das bereitgestellte Bild eine Bildnummer größer

als oder gleich der Inpoint-Bildnummer wird diese Schleife verlassen. Der Stream ist nun korrekt für die Weiterverarbeitung positioniert.

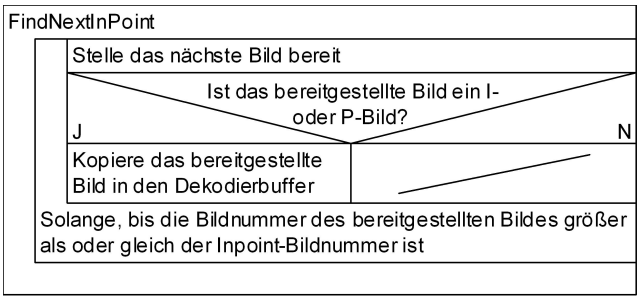

Abbildung 3.10.: Struktogramm der *CScene*-Methode *FindNextInPoint*

3.5.2. *HandleInpoint*

Die Methode *Handle-Inpoint*, ebenfalls integriert in die Klasse *CScene*, wird aufgerufen, wenn, wie im Kapitel 3.5.1 beschrieben, die aktuelle Bildnummer größer als oder gleich der Inpoint-Bildnummer ist. Folgende Situationen können nun vorliegen:

Variante Eins: Die aktuelle Bildnummer ist gleich der Inpoint-Bildnummer und ein I-Bild Diese Situation stellt den einfachsten aller Fälle dar. Das Bild ist ein I-Bild, es kann also auf eine Neukodierung der Inpoint-Umgebung verzichtet werden.

Variante Zwei: Das aktuelle Bild besitzt eine Bildnummer größer als oder gleich der Inpoint-Bildnummer und ist ein P-Bild Hier ist eine Neukodierung unumgänglich. Ist die aktuelle Bildnummer gleich der Inpoint-Bildnummer, ist das nächste Bild wahrscheinlich ein B-Bild, welches eine Bildnummer kleiner der Inpoint-Bildnummer besitzt. Es ist also darauf zu achten, dass jedes einzelne Bild auf seinen Wiedergabezeitpunkt hin überprüft und gegebenenfalls nicht kopiert wird. Ist die aktuelle Bildnummer größer als die Inpoint-Bildnummer, ist das Inpoint-Bild ein B-Bild. Das in der Abspielreihenfolge spätere nächste P-Bild liegt in diesem Moment vor.

Variante Drei: Die aktuelle Bildnummer ist größer als die Inpoint-Bildnummer und ein I-Bild Diese Situation setzt voraus, dass es sich um einen *Offenen GOP* handelt. Die neu zu kodierende Inpointumgebung besteht nur aus B-Bildern. Sie liegen in der Speicherreihenfolge hinter dem I-Bild, auf welches sie sich vorwärts beziehen. Außerdem beziehen sie sich rückwärts auf das letzte P-Bild, welches nicht mitkopiert wird. Diese Situation ist im Ansatz ähnlich zur Variante 2, dennoch zwingen grundlegende Unterschiede zu getrennten Lösungsansätzen und somit Programmzweigen, wie nachfolgend erklärt.

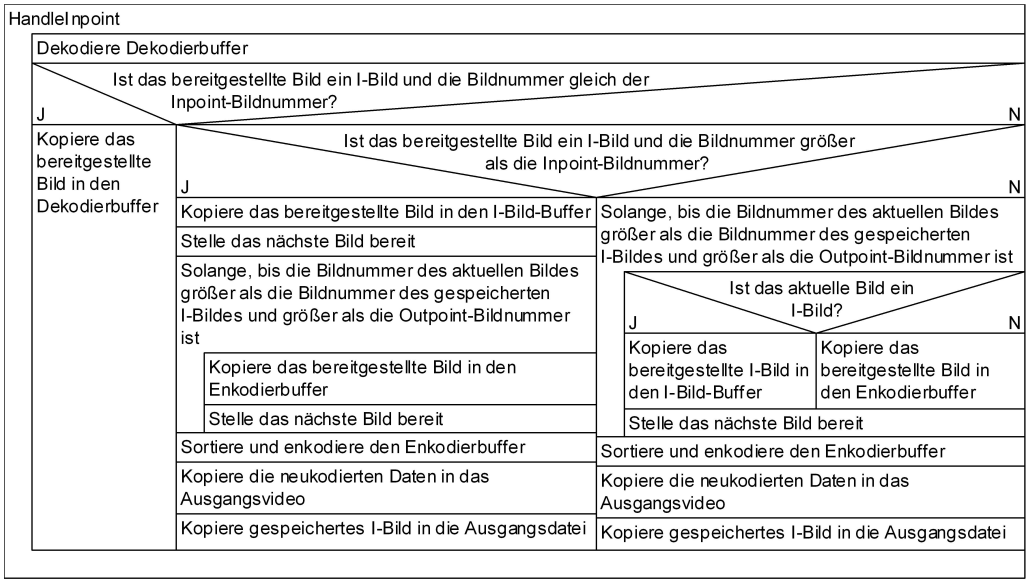

Abbildung 3.11.: Struktogramm der *CScene*-Methode *HandleInpoint*

HandleInpoint folgt dem in Tabelle 3.11 gezeigten Struktogramm.
Zu Beginn wird der Dekodierbuffer dekodiert (siehe Kapitel 3.4.4). Nun geht es zur Wahl der Inpointvariante. Wird die Frage „Ist das bereitgestellte Bild ein I-Bild und die Bildnummer gleich der Inpoint-Bildnummer“ bejaht, muss der Inpoint nach Variante Eins behandelt werden.

Variante Eins (linke Spalte im Flussdiagramm): Da wie beschrieben auf eine Neukodierung verzichtet werden kann, wird an dieser Stelle das aktuelle Bild 1:1 in die Ausgangsdatei geschrieben, die Methode wird beendet.

Handelt es sich nicht um die Inpointvariante Eins, identifiziert die nächste Frage, ob es sich um die Inpointvariante Zwei oder Drei handelt. Diese fragt, ob das aktuelle Bild ein I-Bild ist und die Bildnummer größer als die Inpoint-Bildnummer ist. Ist dies nicht der Fall, wird der Inpoint nach Variante Zwei behandelt.

Variante Zwei (rechte Spalte im Flussdiagramm): Für diese Variante ist eine Neukodierung notwendig. Es folgt die Haupt-While-Schleife mit der Schleifenbedingung „Solange bis die aktuelle Bildnummer größer als die gespeicherte I-Bildnummer und größer als die Outpoint-Bildnummer ist“. Ist das aktuelle Bild ein I-Bild, wird es in den I-Bild-Buffer kopiert (siehe Kapitel 3.4.4), ansonsten wird das Bild dekodiert und in unkomprimierter Form in den Enkodierbuffer (siehe Kapitel 3.4.4) kopiert. Anschließend wird das nächste Bild gelesen. Ist die Haupt-Schleifenbedingung nicht mehr erfüllt, wird der Enkodierbuffer enkodiert und die neukodierten Daten werden in das Ausgangsvideo geschrieben. Anschließend wird das zwischengespeicherte I-Bild ebenfalls in den Ausgangsdatenstrom geschrieben.

Nun folgt die Erläuterung des Struktogramms für die Inpointvariante Drei, welche durchlaufen wird, wenn das aktuelle Bild ein I-Bild ist und eine Bildnummer größer als die Inpoint-Bildnummer besitzt.

Variante Drei (mittlere Spalte im Flussdiagramm): Hier wird von der Tatsache ausgegangen, dass das aktuelle Bild ein I-Bild ist und nun eine beliebige Anzahl von B-Bildern mit einer kleineren Bildnummer als die aktuelle I-Bildnummer folgen. Daher wird im ersten Schritt das aktuelle I-Bild in den I-Bild-Buffer (siehe Kapitel 3.4.4) kopiert. Das nächste Bild wird gelesen. Nun folgt die Haupt-While-Schleife mit der Schleifenbedingung „Solange bis die aktuelle Bildnummer größer als die gespeicherte I-Bildnummer und größer als die Outpoint-Bildnummer ist“. In dieser Schleife wird das aktuelle Bild dekodiert und in den Enkodierbuffer kopiert. Das nächste Bild aus dem Quellvideo wird eingelesen. Auch hier wird, wenn die Haupt-Schleifenbedingung nicht mehr erfüllt ist, der Enkodierbuffer enkodiert und die neukodierten Daten in das Ausgangsvideo geschrieben. Anschließend wird

ebenfalls das zwischengespeicherte I-Bild in den Ausgangsdatenstrom geschrieben.

Die Inpointumgebung ist nun gültig behandelt, es kann nun soweit es die EDL verlangt mit der 1:1-Kopie der weiteren Bilder begonnen werden.

3.5.3. *HandleRestOfStream*

HandleRestOfStream ist ebenfalls integriert in die Klasse *CScene* und folgt dem in Abbildung 3.12 gezeigten Struktogramm.

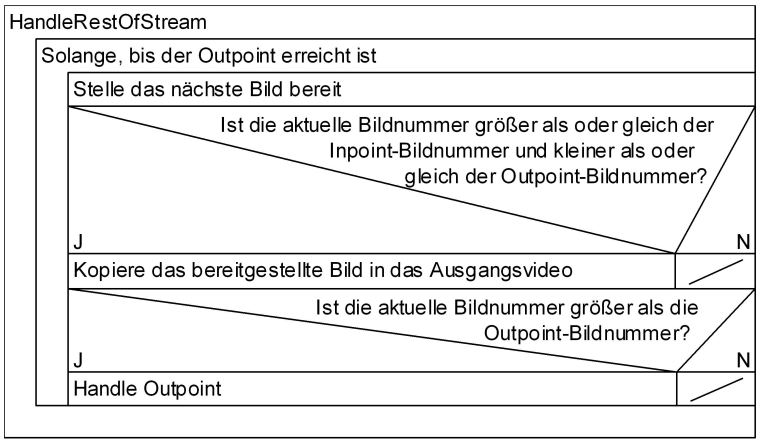

Abbildung 3.12.: Struktogramm der *CScene*-Methode *HandleRestOfStream*

Es wird solange das nächste Bild gelesen und in den Ausgangsstream kopiert, bis entweder das Outpointbild direkt geschrieben oder aber die Bildnummer des aktuellen Bildes größer als die Outpoint-Bildnummer ist, und somit in die Methode „Handle Outpoint“ verzweigt wird. Dies geschieht aus folgendem Grund: Wie in Kapitel 2.2 beschrieben kann ein gültiges Outpointbild durch entweder ein I- oder P-Bild gebildet werden. Besitzt das aktuelle Bild eine Bildnummer gleich der Outpointbildnummer, ist es ein I- oder P-Bild, es können nun verbleibende B-Bilder folgen, welche im Anschluss an das Ende des Ausgangsvideos kopiert werden müssen. Dieser Fall kommt ohne eine Neukodierung aus.
Anders wenn die aktuelle Bildnummer größer als die Outpoint-Bildnummer ist. Dann wird das Outpointbild ein Bild sein, welches sich auf dieses aktuell

vorliegende Bild vorwärts bezieht, also ein B-Bild. Es ist dann eine Neukodierung der Outpointumgebung notwendig.

3.5.4. *HandleOutpoint*

Diese Methode, ebenfalls integriert in die Klasse *CScene*, ist erforderlich bei Neukodierungsbedarf der Outpointumgebung, Abbildung 3.13 zeigt das entsprechende Struktogramm.

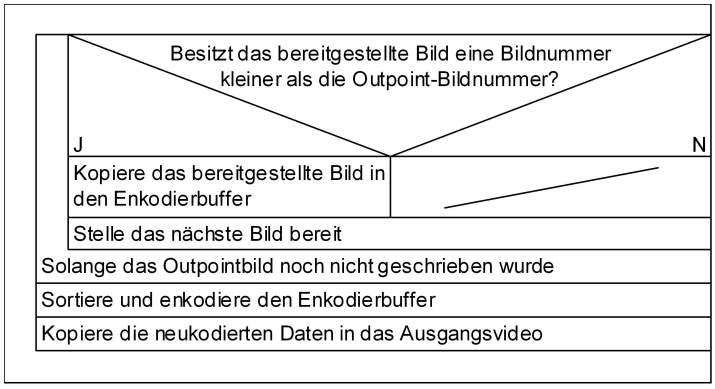

Abbildung 3.13.: Struktogramm der *CScene*-Methode *HandleOutpoint*

Wird diese Methode aufgerufen, beginnt die Haupt-Do-While-Schleife mit der Schleifenbedingung „Solange das Outpointbild noch nicht geschrieben wurde“. In dieser Schleife wird das nächste Bild aus dem Quellvideo gelesen, dekodiert und in den Enkodierbuffer geschrieben.
Wenn das Outpointbild in den Enkodierbuffer geschrieben wurde, wird die Schleife beendet, die sich im Enkodierbuffer befindenen Bilder werden neu kodiert und die neukodierten Daten werden in das Ausgangsvideo geschrieben. Die Szene ist gültig abgeschlossen.

3.5.5. *CopyFrame*

Diese in Abbildung 3.14 gezeigte Methode gehört zur Klasse *COutput*. Ihr wird ein Bild übergeben, welches in den Ausgangsdatenstrom kopiert wird. Das Bild kann sowohl ein progressives als auch ein Halbbild sein. Es kann

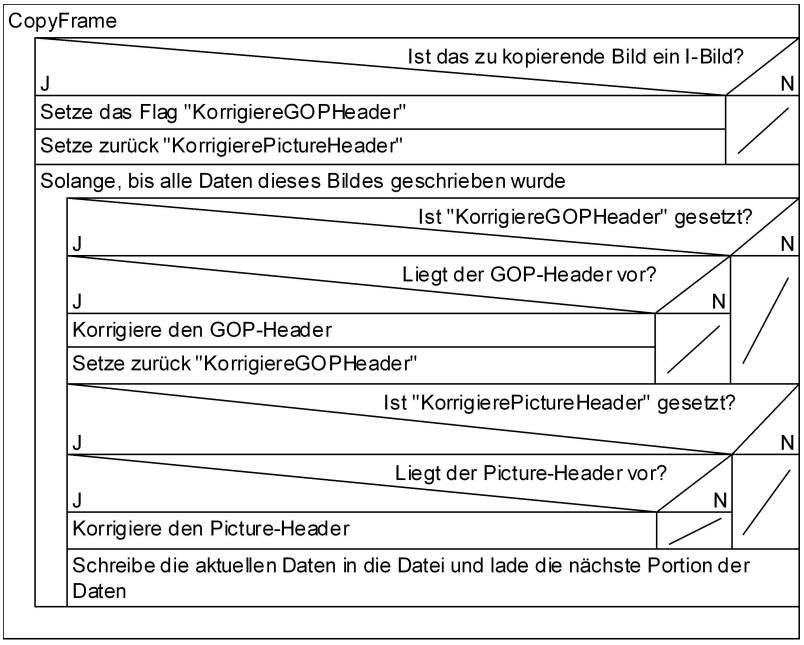

Abbildung 3.14.: Struktogramm der *COutput*-Methode *CopyFrame*

entweder direkt als 1:1-Kopie aus dem Stream oder aus dem Enkoder kommen.
Bereits beschriebene Modifikationen werden hier durchgeführt, es handelt sich dabei um die *Picture-Header*-Korrektur, welche die *Temporal Reference* modifiziert, sowie die *GOP-Header*-Korrektur, welche die *GOP*-Parameter *Closed GOP* und den Timecode mit den aktuell vorliegenden Informationen überschreibt.
Sind alle zu editierenden Header gefunden, sorgt die Programmstruktur dafür, dass nicht weiter nach entsprechenden Headern gesucht wird, sondern der Rest der Daten unmittelbar in das Ausgangsvideo geschrieben werden, sodass die nächsten Arbeitsschritte eingeleitet werden können.

3.6. Geeignete Testmethoden

Um zu gewährleisten, dass die Software einwandfrei funktioniert, ist sicherzustellen, dass sämtliche von der Software in den bestehenden Stream vollzogenen Eingriffe zum einen das gewünschte Schnittergebnis liefern und zum anderen die Gültigkeit des Streams nicht aufheben.

Testmethoden sind grundsätzlich in zwei Testrichtungen aufzuteilen:

- Überprüfung des Schnittergebnisses bezüglich der Korrektheit aller gesetzten In- und Outpoints
- Überprüfung der Gültigkeit des Streams

3.6.1. Überprüfung des Schnittergebnisses bezüglich der Korrektheit aller gesetzten In- und Outpoints sowie Sichtkontrolle

Zur Überprüfung des Schnittergebnisses bezüglich der korrekten In- und Outpoints wird das den IPP-UMC-Klassen beigelegte Beispielprogramm „Simple Player“ verwendet. Dieses liest Videodateien sämtlicher Formate und rendert sie auf dem Bildschirm. Durch Modifizierung des Quelltextes wurde dieses Programm um die Ausgabe der Bildnummer in der Eingabeaufforderung ergänzt. Die Bildnummer wird berechnet aus der Bildwiedergabezeit geteilt durch die Bildwiederholrate. Durch die ausgegebene Bildnummer ist ersichtlich, ob die Modifikation der *Temporal Reference* bei 1:1-Kopien bzw. der Neukodierung und anschließender 1:1-Kopie korrekt erfolgte. Keine fortlaufende Bildnummer deutet auf fehlerhafte *Temporal Reference*-Modifkation hin.
Zusätzlich zur Bildnummer wird vom Programm der Frametyp des jeweiligen Bildes ausgeben. Mit dieser Maßnahme kann der korrekte Einsatz der I-, P- und B-Bilder kontrolliert werden, jedoch spiegeln sich Fehler in diesem Bereich auch deutlich im wiedergegebenen Bild wieder.
Aus dem gerenderten Videobild lassen sich die meisten Aussagen treffen, ob der Schnitt geglückt ist. An ihm lassen sich folgende Fehlertypen ablesen:

- Bildsortierfehler
- Enkodierfehler
- Farbfehler

Durch Videofiles mit fortlaufend nummerierten Bildern im Videobild ist man zusätzlich in der Lage, den Schnitt bezüglich der korrekt gesetzten In- und Outpoints zu kontrollieren.

3.6.2. Überprüfung der Gültigkeit des Streams

Bei Annahme einer korrekten Arbeitsweise der UMC-Klassen ist zur Überprüfung der Gültigkeit des Ausgangsvideos ausreichend, ihn auf den Ebenen zu betrachten, in welchen der Stream auch bearbeitet wurde. Es interessieren also: Die Gültigkeit der Sequenz und der *Sequence-Header*, die Gültigkeit der *GOP´s* und der *GOP-Header* und die Gültigkeit der *Pictures* und der *Picture-Header*.
Alle weiteren Ebenen kommen folglich nicht für die Fehlersuche in Frage, da ausschließlich auf genannten Ebenen editiert wurde.

Die Gültigkeit der Sequenz und des *Sequence-Headers*

Jede Sequenz beginnt mit dem *Sequence-Header*, beginnend mit dem *Sequence-Start-Code*, optional gefolgt von der *Sequence-Extension*. *Sequence-Header* können in regelmäßigen Abständen vor I- oder P-Bildern wiederkehren, dann sollten jedoch auch sämtliche Parameter denen des ersten *Sequence-Headers* entsprechen. Die Sequenz wird abgeschlossen durch den *Sequence-End-Code*.
Aufgrund dessen, dass der *Sequence-Header* nicht modifiziert, sondern nur kopiert wird, ist es nicht notwendig, sich mit dessen Inhalt zu beschäftigen, sondern ausschließlich mit dessen Vorhandensein. Es ist also zu kontrollieren, ob der *Sequence-Header* am Anfang einmal, und, falls der *Sequence-Header* im Original wiederkehrend ist, der *Sequence-Header* auch im Ergebnis an gültigen Stellen vorhanden ist. Desweiteren ist es wichtig, den *Sequence-End-Code* auf das nur einmalige Vorkommen nur am Ende der Sequenz zu überprüfen.
Eine geeignete Methode zur Überprüfung der Ergebnis-Datei stellt die Verwendung einer MPEG2-Analysesoftware dar. Unter anderem die Software MTS4EA der Version 4.2.0.2 aus dem Hause Tektronix[5] verspricht die vollständige Analyse einer Vielzahl digitaler Video-Formate. Eine derartige Software wird von VCS für diese Arbeit jedoch nicht bereitgestellt, daher wird hier die Ergebnis-Datei mittels der Hexadezimal-Darstellung des Microsoft Visual Studio C++ auf die beschriebenen Eigenschaften hin überprüft.

Die Gültigkeit der *GOP´s* und der *GOP-Header*

GOP-Header sind optionale Header und kommen, wenn vorhanden, unmittelbar vor kodierten I-Frames vor. Diese Header werden regelmäßig von dem Programm in folgender Art und Weise modifiziert: Die Timecode-Informationen werden mit dem aktuellen Timecode überschreiben, sowie das Flag *Closed-Gop* wird angepasst. Diese beiden Parameter werden zum einen zur Kontrolle beim Erstellen der Ausgabedatei auf dem Bildschirm ausgegeben, bzw. können im Simple-Player optional mit ausgegeben und somit kontrolliert werden.

Die Gültigkeit der *Pictures* und der *Picture-Header*

Bei jedem Schnitt, bei dem ein I-Bild als Inpoint dient, bzw. die Inpoint-Umgebung neu kodiert und anschließend ein I-Bild angesetzt wurde, ist bei Verwendung von *Offenen Gop´s* eine Modifikation der *Temporal-Reference* notwendig. Das Programm korrigiert im gesamten *GOP* sämtliche *Temporal References*. Kontrolle über diesen Vorgang hat man zum einen durch die Ausgabe beim Erstellen der Ausgabedatei, sowie beim Betrachten der Bildnummern bei dem Abspielen der Ausgabedatei im Simple Player. Alle weitere Komponenten des *Picture-Headers* bleiben unangetastet – bei Verwendung der UMC-Klassen werden automatisch alle *Picture-Header* mitkopiert. Die Kontrolle des Vorhandenseins ist demnach nicht notwendig.

4. Ergebnis

4.1. Test der Software

4.1.1. Aufbau und Eigenschaften der Testvideos

Zur Identifizierung jedes einzelnen Bildes ist es sinnvoll, eine Testsequenz mit Bildnummernzähler zu generieren. Desweiteren wurden, da auch mehrere Videofiles als Quelle gleichzeitig handhabbar sein sollten, vier von diesen Testsequenzen mit jeweils einer anderen Bildnummer-Hintergrundfarbe erzeugt.
Diese vier Testvideos dienen als Grundlage für folgende dem MPEG2-Standard entsprechenden Videos:

Standardparameter:	IBBPBB...
	CBR=8 Mbit/s
	MP@ML 4:2:0
	SequenceHeader vor jedem GOP
_[farbe]_prog.m2v:	Progressiv
_[farbe]_field_bot.m2v:	interlaced fieldpictures, bottom field first
_[farbe]_field_top.m2v:	interlaced fieldpictures, upper field first
_[farbe]_frame_bot.m2v:	interlaced framepictures, bottom field first,
	CBR = 6 Mbit/s
_[farbe]_frame_top.m2v:	interlaced framepictures, upper field first

Sowie in jeweils einfacher Ausführung mit den gleichen Standardparametern:

_longgop.m2v:	Progressiv, Gop-Länge von 30,
	Geschlossene GOP´s
_hphl_422.m2v:	HP@HL 4:2:2

Sowie das dem IMX-System-entsprechende Testvideo ohne eingeblendete Bildnummern:

_hires_video.m2v:	XDCAM IMX-entsprechendes MPEG2-Video 4:2:2 intraframe interlaced framepictures, upper field first CBR = 50 Mbit/s

Die Struktur der Videos mit Ausnahme des Intraframe-Videos ist im Anhang beigefügt.

4.1.2. Erläuterung des Testprotokolls

Das Testprotokoll soll unter Berücksichtigung aller oben genannten Punkte die Arbeitsweise des Programmes überprüfen. Es bezieht sich auf spezielle Test-EDLs, die im Folgenden erläutert werden.
Das Testprotokoll ist unterteilt in 6 Punkte:

1. Test aller Schnittmöglichkeiten bei Progressive Frames
2. Test aller Schnittmöglichkeiten bei Interlaced Framepictures UFF
3. Test aller Schnittmöglichkeiten bei Interlaced Fieldpictures UFF
4. Test aller Schnittmöglichkeiten mit langer *GOP*-Struktur
5. Test aller Schnittmöglichkeiten mit dem Intraframe-Video
6. Test der Ausnahmefälle

In \ Out	**I-Bild**	**P-Bild**	**B-Bild**
I-Bild	1	2	2
P-Bild	1	2	2
B-Bild	1	2	2
critical B-Bild	1	1	2

Tabelle 4.1.: Alle Schnittmöglichkeiten

Die Test-EDL soll dem Programm alle Schnittvarianten abverlangen. Dazu ist es wichtig, in dieser EDL sämtliche Möglichkeiten der In- und Outpoints durchzuspielen. Tabelle 4.1 zeigt alle Möglichkeiten von In- bzw. Outpoints. Die als „critical B“-bezeichneten Bilder stehen für die *GOP*-übergreifenden B-Bilder bei *Offenen GOP´s*.
Die mit „2“ gekennzeichneten Kombinationen sind *GOP*-intern sowie *GOP*-übergreifend möglich.

Szene	**In / Out**
1	I / I
2	I / P
3	I / P (GOP-intern)
4	I / B
5	I / B (GOP-intern)
6	P / I
7	P / P
8	P / P (GOP-intern)
9	P / B
10	P / B (GOP-intern)
11	B / I
12	B / P
13	B / P (GOP-intern)
14	B / B
15	B / B (GOP-intern)
16	critical B / I
17	critical B / P
18	critical B / B
19	critical B / B (GOP-intern)
20	einzelnes I-Bild
21	einzelnes P-Bild
22	einzelnes B-Bild
23	Outpoint ist gleich
24	wie Inpoint (I)
25	Outpoint ist gleich
26	wie Inpoint (P)
27	Outpoint ist gleich
28	wie Inpoint (B)
29	Outpoint ist gleich
30	wie Inpoint (critical B)
31	Outpoint ist letztes Bild (GOP-intern)
32	Outpoint ist letztes Bild (GOP-übergreifend)
33	Outpoint ist vorletztes Bild (GOP-intern)
34	Outpoint ist vorletztes Bild (GOP-übergreifend)
35	Standardszene

Tabelle 4.2.: Theoretische EDL

Desweiteren werden folgende Fälle getestet:

- Letzter Outpoint gleich wie der aktuelle Inpoint (I)
- Letzter Outpoint gleich wie der aktuelle Inpoint (P)
- Letzter Outpoint gleich wie der aktuelle Inpoint (B)
- Letzter Outpoint gleich wie der aktuelle Inpoint (critical B)
- Outpoint ist letztes Bild in dem Quellvideo (2)
- Outpoint ist vorletztes Bild in dem Quellvideo (2)

Aus diesen Schnittmöglichkeiten ergibt sich die in Tabelle 4.2 gezeigte theoretische EDL.
Die Test-EDL entsteht mit Hilfe des Aufbaus der einzelnen Videos, indem entsprechende Bilder als In- bzw. als Outpoints gewählt werden. Stellvertretend für die Testprotokollpunkte 1-3 ist nachfolgend das Testprotokoll für den Test aller Schnittmöglichkeiten bei progressiven Bildern dargestellt.

Test-EDL Progressive Frames:

```
start sourcefiles
d:fokkenvideosamples_blau_prog.m2v
d:fokkenvideosamples_rot_prog.m2v
d:fokkenvideosamples_gelb_prog.m2v
d:fokkenvideosamples_braun_prog.m2v
end sourcefiles

start edllines
1    0    17   32    //I-I
2    1    152  176   //I-P
3    2    257  262   //I-P(GOP-intern)
4    0    348  367   //I-B
5    3    62   75    //I-B(GOP-intern)
6    2    248  257   //P-I
7    2    306  330   //P-P
8    1    143  146   //P-P(GOP-intern)
9    2    38   52    //P-B
10   0    176  180   //P-B(GOP-intern)
11   1    276  288   //B-I
12   0    325  342   //B-P
13   0    157  164   //B-P(GOP-intern)
14   1    298  313   //B-B
15   3    352  353   //B-B(GOP-intern)
16   2    346  363   //critical B-I
17   2    331  336   //critical B-P
18   1    210  216   //critical B-B
19   0    165  166   //critical B-B(GOP-intern)
20   1    303  303   //einzelnes I-Frame
```

```
21   2    98   98    //einzelnes P-Frame
22   3    368  368   //einzelnes B-Frame
23   2    33   47    //letzter Outpoint gleich
24   2    47   60    //wie aktueller Inpoint(GOP-intern)
25   1    140  146   //letzter Outpoint gleich
26   1    146  155   //wie aktueller Inpoint (P)
27   0    240  250   //letzter Outpoint gleich
28   0    250  260   //wie aktueller Inpoint (B)
29   1    280  286   //letzter Outpoint gleich
30   1    286  290   //wie aktueller Inpoint (critical B)
31   1    355  374   //Outpoint ist letztes Frame
32   1    364  374   //Outpoint ist letztes Frame (GOP-intern)
33   2    355  373   //Outpoint ist vorletztes Frame
34   2    364  373   //Outpoint ist vorletztes Frame (GOP-intern)
35   2    0    20    //Standardszene
end edllines
```

Szene	Bildsortier-fehler	Encodier-fehler	Farb-fehler	Inkorrekter In- oder Outpoint	Inkorrekte Darstell-ung der Bildnummer
1					
2					
3					
...					
35					
Sequence-Header korrekt verwendet					
Sequence-End-Code korrekt verwendet					

Tabelle 4.3.: Testprotokoll der Testkontrollpunkte 1-4

Testkontrollpunkt 4: Test aller Schnittmöglichkeiten bei langen GOP-Strukturen

Das Video „LongGop.m2v“ besteht ausschließlich aus Geschlossenen GOP´s. Daher wird das Testprotokoll um die Schnittmöglichkeiten der GOP-übergreifenden B-Bilder verringert.

Testkontrollpunkt 5: Test aller Schnittmöglichkeiten mit dem Intraframe-Video

Da es sich um ein Intraframe-Video handelt, fällt der Test entsprechend der wegfallenden Schnittmöglichkeiten kürzer aus – desweiteren können bei diesem Video nicht die In- und Outpoints kontrolliert werden, da es keine eingeblendeten Framenummern enthält. Darüber hinaus ist die Erstellung

eines Testvideos mit eingeblendeten Bildnummern mit den von VCS bereitgestellten Hilfsmitteln nicht möglich. Alle weiteren Kriterien werden jedoch überprüft.

Testkontrollpunkt 6: Test der Ausnahmefälle
Es werden nacheinander folgende Fehler in die EDL eingebaut:

- die EDL ist nicht verfügbar
- ein Dateiname ist falsch geschrieben
- ein Inpoint ist größer als der dazugehörige Outpoint
- es wird ein Link benutzt, welcher größer als die Anzahl der gesamt verfügbaren Links ist
- es wird die Datei `_hphl_422.m2v` gemeinsam mit `_[farbe]_prog.m2v` eingetragen
- es wird die Datei `_[farbe]_frame_bot.m2v` gemeinsam mit `_[farbe]_frame_top.m2v` eingetragen
- es wird die Datei `_hires_video.m2v` zusammen mit `_[farbe]_field_bot.m2v` eingetragen

Bei all diesen Ausnahmefällen sollte das Programm mit einem entsprechenden Fehlerhinweis beendet werden. Auch dieses Ergebnis wird in dem Testprotokoll festgehalten.

4.1.3. Test der Software nach dem Testprotokoll

Das gemäß dem Test ausgefüllte Testprotokoll ist zu finden im Anhang. Während dem Test sind keine Unregelmäßigkeiten aufgetreten, die Software hat den Test bestanden.

5. Diskussion des Ergebnisses

5.1. Endergebnis

Die Aufgabenstellung dieser Arbeit sah zum einen die Entwicklung einer Software vor, welche mittels einer EDL und verschiedenen Quellvideos als Eingabe ein fertig geschnittenes Video erstellt, zum anderen sollte diese Software in ihrem Laufzeitverhalten nach technischen Möglichkeiten optimiert werden (vgl. Kapitel 1).

Die Tests haben gezeigt, dass der erste Punkt soeben wiederholter Aufgabenstellung erfüllt ist. Die Software ist in der Lage, aus den Quellvideos gemäß der EDL ein Ausgabevideo zu erstellen. Als Quellvideos kommen ausschließlich MPEG2-konforme Video-Elementarströme in Betracht, der Schnitt von Audiomaterial wird nicht unterstützt.

Wenn möglich wird das Quellvideo auf Bitstromebene 1:1 kopiert, nur in nicht vermeidbaren Fällen wird eine Neukodierung durchgeführt. Im Verlauf dieser Arbeit wurden die theoretischen Hintergründe der Schnittmöglichkeiten innerhalb eines MPEG2-Videodatenstroms ohne Neukodierung beleuchtet.

Im Zuge der Neukodierung werden Bilder korrekt für den Enkoder umsortiert, im Anschluss an die Neukodierung werden diese Daten korrekt in den Ausgangsdatenstrom geschrieben. Auch diese Mechanismen sind innerhalb dieser Arbeit erläutert worden.

MPEG2-spezifische Probleme in Verbindung mit dem Schnitt des Quellvideomaterials auf Bitstromebene wurden betrachtet und im Endergebnis

in Form der entwickelten Software umgesetzt, so werden z.B. verschiedene Flags aus 1:1-kopierten Teilstücken aus dem Quellvideo ausgelesen und während des Kopiervorganges modifiziert.

Die vom MPEG2-System gegebene Möglichkeit zum Schreiben eines Timecodes in den Datenstrom wird in dieser Software genutzt, die Notwendigkeit und Theorie wurde nahegebracht.

Der zweite Punkt oben genannter Aufgabenstellung, die Optimierung des Laufzeitverhaltens der Software nach technischen Möglichkeiten wurde ebenfalls erfüllt. So arbeitet diese Software sehr effizient bei Betrachtung notwendiger De- und Enkodiervorgänge, da innerhalb der Arbeitsgänge der Software kaum mehr Dekodiervorgänge vollzogen werden, als notwendigerweise durchzuführen sind. Weiter aufzuführen ist der sehr effiziente Mechanismus zur Rücksortierung der Bilder innerhalb des Enkodierbuffers, sowie die Methodik hinter dem Dekodierbuffer.
Dazu sind die Mechanismen des Suchens der zu kopierenden Bilder hervorzuheben, wobei hier die Methodik zur Wiederaufnahme des Suchens ab der zuletzt benutzten Stelle innerhalb eines Quellvideos, sowie zur Erstellung eines Index mit gespeicherten absoluten Positionen der Bilder zum Positionieren des Lesezeigers im Stream zu erwähnen sind. Diese Maßnahmen bringen einen relativ großen Effizienzgewinn.

5.2. Einschränkungen der Software

Wie in Kapitel 5.1 beschrieben wurde das gesteckte Ziel erreicht. Das derzeitige Funktionsangebot der entwickelten Software ist jedoch beschränkt, es gibt über das gesetzte Ziel hinaus noch viele mögliche Ausbaustufen.
Einige wünschenswerte Möglichkeiten zum Ausbau der Software wären z.B. eine Erweiterung des Kompatibilitätsumfanges bezüglich der unterstützten Kompressionsformate der Quellvideos auf MPEG4- und DV-Videomaterial und das Behandeln von Programm- und Transportströmen zusätzlich zu den Videoelementarströmen sowie eine Integration des Audioschnittes (vgl. Kapitel 6).

5.3. Notwendigkeit und Alternativen

Die innerhalb dieser Arbeit entstandene Software begründet ihre Notwendigkeit in der Lösung der Aufgabe, Videomaterial nach einer EDL effizient zu schneiden (vgl. Kapitel 1).
Zur Lösung dieser Aufgabe gibt es neben der in dieser Arbeit vorgestellten Lösung natürlich weitere Lösungen. Es besteht die Möglichkeit, bereits entwickelte Software zu erwerben, als kostenloses Beispiel wäre aufzuführen das frei erhältliche Produkt „Cuttermaran“ der Version 1.67 [6], welches ähnliche Funktionen wie die entwickelte Software bereitstellt. Jedoch ist diese Software als Benutzeranwendung anstatt als Serveranwendung programmiert. Weiter gibt es auch professionelle nicht frei verfügbare Software zur Erfüllung der Aufgabe.
Softwareeigenentwicklungen sind zumeist sehr teuer. Der Grund für die VCS AG, trotzdem die Software in Eigenarbeit zu entwickeln, war zum einen die Möglichkeit, die Software den gewünschten Bedingungen und Funktionen genau anpassen und zu gegebener Zeit auch erweitern zu können. Zum anderen besteht für gekaufte Software die Gefahr, dass vom jeweiligen Hersteller ein Support auch in der Zukunft nicht zwingend sichergestellt ist.

6. Ausblick

Dieser Ausblick beschäftigt sich mit Möglichkeiten zum Ausbau der im Rahmen dieser Arbeit entstandenen Software. Diese Software ist in der letzten Version im Stande, MPEG2-Videoelementarströme nach einer EDL hart zu schneiden. Es können mehrere Videos als Quelle dienen.
Ausbaustufen können in folgenden Richtungen den Funktionalitätsumfang dieses Programms aufweiten:

- Der zum Videomaterial zugehörige Ton kann ebenfalls mitgeschnitten werden.
- Das bisher unterstützte Videoformat MPEG2 kann durch weitere ergänzt werden (MPEG1, MPEG4, H261, H263, H264, DV).
- Programmströme, Transportströme sowie Elementarströme können gleichermaßen verarbeitet werden.
- Über die harten Schnitte hinaus können weitere Überblendarten ermöglicht werden (weiche Überblendung, Blende über Schwarz/Weiss, Ein- bzw. Ausblendung)
- Es können weitere Effekte auf das Audio- und/oder Videomaterial anwendbar sein (Rückwärtslauf, Standbild, Geschwindigkeitsänderung, Fehlererkennung)
- Es können Grafiken z. B. zur Kennung des Videomaterials in den Videodatenstrom eingerechnet werden.

Abkürzungen

Abkürzung	**Beschreibung**
EDL	Edit Decision List, Schnittliste
GOP	Group Of Pictures
Highres	High Resolution, hochauflösend
LFF	Lower Field First
Lowres	Low Resolution, niedrigauflösend
MBS	Media Broadcast Solutions
MPEG	Motion Picture Expert Group
UFF	Upper Field First

Literaturverzeichnis

[1] MPEG2 (Video) ISO/IEC 12818-2: 1995 E

[2] MPEG1 (Video) ISO/IEC 11172-2

[3] Intel, Unified Media Data Classes - Reference Manual, Doc.-Nr.: 306061-003US

[4] Tilo Strutz, Bilddatenkompression 3. Auflage (2005)

[5] Homepage Tektronix, Stand 19. Juni 2006, http://www.tektronix.com/products/video_test/mts4ea/index.html

[6] Homepage Cutterman (T. Arnold), Stand 19. Juni 2006, http://www.cuttermaran.de

Abbildungsverzeichnis

Tabellenverzeichnis

A. Inhalt der CD-R

Diplomarbeit

Verzeichnis: /diplomarbeit
Hier ist das PDF-Dokument der Diplomarbeit in hoher Auflösung abgelegt.

- Matthias_Fokken_-_Diplomarbeit_VCS.pdf

Online-Quellen

Verzeichnis: /quellen
Die im Literaturverzeichnis angegebenen Online-Quellen sind ebenfalls auf der beigelegten CD-ROM enthalten. Abgespeichert wurden von einer Internet-Seite jeweils nur die Web-Seiten, von denen Informationen entnommen wurden.

- http://www.tektronix.com/products/video_test/mts4ea/index.html
  Zugriff: 27. Juni 2006
- http://www.cuttermaran.de
  Zugriff: 27. Juni 2006

Grafiken

Verzeichnis: /bilder/grafiken

- hauptobjektstruktur.jpg – Allgemeine Objektstruktur der entwickelten Software
- ibild-buffer.jpg – Zusammenhänge der unterschiedlichen verwendeten Buffer

- klassen.jpg – Die Klassen und ihre Methoden und Eigenschaften
- komunikation.jpg – Daten- und Informationsaustausch
- manager.jpg – Workflow des Managers
- mediaundvideodaten.jpg – Datenfluss der komprimierten und unkomprimierten Videodaten
- modifikationen.jpg – Wirkungsweise der *Temporal Reference*-Modifikation
- Theorie0.jpg – *GOP*-Strukturen
- Theorie1.jpg – Beispielschnittumgebung ohne Neukodierungsbedarf
- Theorie2.jpg – Beispielschnittumgebung mit Neukodierungsbedarf
- Theorie3.jpg – Behandlung der Inpointumgebung bei *Offenen GOP´s*
- Theorie4.jpg – Behandlung einer Outpointumgebung mit Neukodierungsbedarf
- umc_workflow.jpg – Arbeitsweise der Intel-UMC-Klassen

Struktogramme

Verzeichnis: /bilder/struktogramme

- MeTCCalculation.jpg – Berechnung des Timecodes
- StCopyFrame.jpg – Kopieren von Videodaten in den Ausgangsdatenstrom
- StFindNextInPoint.jpg – Suchmechanismus zum Finden des Inpoints
- StHandleInpoint.jpg – Behandlung des Inpoints
- StHandleRestOfStream.jpg – Behandlung der verbleibenden zu behandelnden Bilder
- TeHe1.jpg – Logik für Schnittumgebungen ohne Neukodierungsbedarf

- TeHe2.jpg – Logik für Schnittumgebungen mit Neukodierungsbedarf
- TeHe3.jpg – Logik zur geeigneten Behandlung der Inpointumgebung
- TeHe4.jpg – Logik zur geeigneten Behandlung der Inpointumgebung
- TeHe5.jpg – Logik zur geeigneten Behandlung der Outpointumgebung

Testvideos

Verzeichnis: /videos

- _blau_field_bot.m2v, _braun_field_bot.m2v, _gelb_field_bot.m2v, _rot_field_bot.m2v
- _blau_field_top.m2v, _braun_field_top.m2v, _gelb_field_top.m2v, _rot_field_top.m2v
- _blau_frame_bot.m2v, _braun_frame_bot.m2v, _gelb_frame_bot.m2v, _rot_frame_bot.m2v
- _blau_frame_top.m2v, _braun_frame_top.m2v, _gelb_frame_top.m2v, _rot_frame_top.m2v
- _blau_prog.m2v, _braun_prog.m2v, _gelb_prog.m2v, _rot_prog.m2v
- _HPHL_422.m2v, _LongGop.m2v, _hires_video.m2v

Test-EDL´s

Verzeichnis: /edl

- EDL_1.txt – Zum Test aller Schnittmöglichkeiten bei Progressive Frames
- EDL_2.txt – Zum Test aller Schnittmöglichkeiten bei Interlaced Framepictures UFF
- EDL_3.txt – Zum Test aller Schnittmöglichkeiten bei Interlaced Fieldpictures UFF

- EDL_4.txt – Zum Test aller Schnittmöglichkeiten mit langer *GOP*-Struktur
- EDL_5.txt – Zum Test aller Schnittmöglichkeiten mit dem Intraframe-Video
- EDL_6[b-g].txt – Zum Test der Ausnahmefälle

B. Aufbau der Testvideos

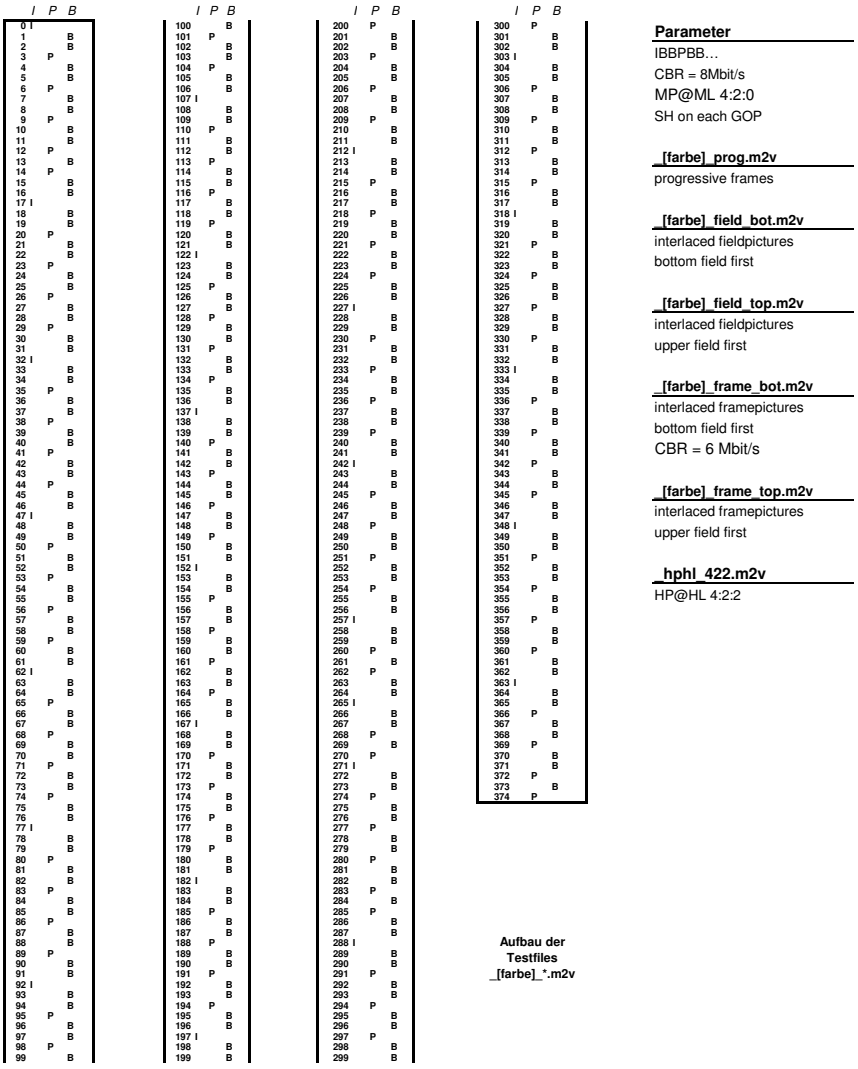

Abbildung B.1.: Aufbau der Testvideofiles „_[farbe]_*.m2v“

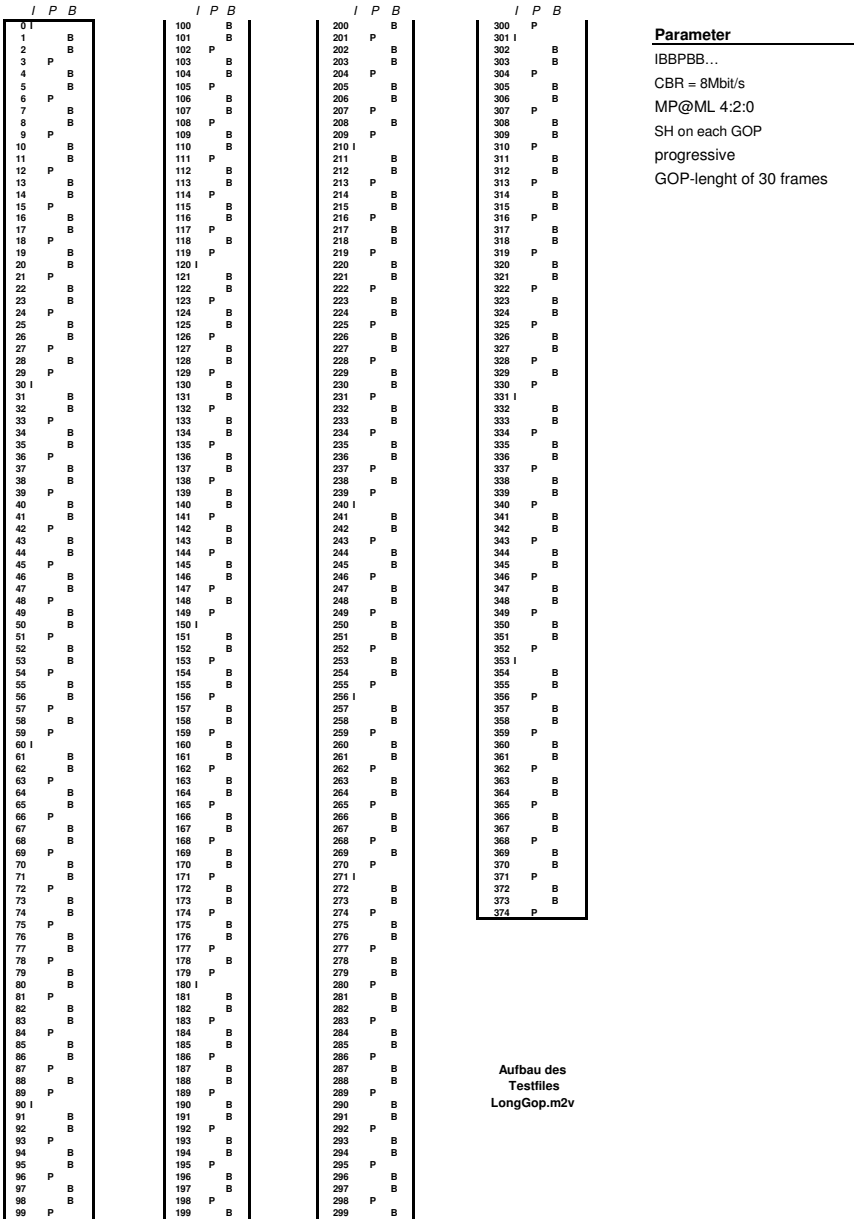

Abbildung B.2.: Aufbau des Testvideofiles „LongGop.m2v“

C. Testprotokolle

Szene	Bildsortier-fehler	Encodier-fehler	Farb-fehler	Inkorrekter In- oder Outpoint	Inkorrekte Darstell-ung der Bildnummer
1	–	–	–	–	–
2	–	–	–	–	–
3	–	–	–	–	–
4	–	–	–	–	–
5	–	–	–	–	–
6	–	–	–	–	–
7	–	–	–	–	–
8	–	–	–	–	–
9	–	–	–	–	–
10	–	–	–	–	–
11	–	–	–	–	–
12	–	–	–	–	–
13	–	–	–	–	–
14	–	–	–	–	–
15	–	–	–	–	–
16	–	–	–	–	–
17	–	–	–	–	–
18	–	–	–	–	–
19	–	–	–	–	–
20	–	–	–	–	–
21	–	–	–	–	–
22	–	–	–	–	–
23	–	–	–	–	–
24	–	–	–	–	–
25	–	–	–	–	–
26	–	–	–	–	–
27	–	–	–	–	–
28	–	–	–	–	–
29	–	–	–	–	–
30	–	–	–	–	–
31	–	–	–	–	–
32	–	–	–	–	–
33	–	–	–	–	–
34	–	–	–	–	–
35	–	–	–	–	–
Sequence-Header korrekt verwendet				ok	
Sequence-End-Code korrekt verwendet				ok	

Tabelle C.1.: Testprotokoll nach Punkt 1: Test aller Schnittmöglichkeiten bei Progressive Frames

Szene	Bildsortier-fehler	Encodier-fehler	Farb-fehler	Inkorrekter In- oder Outpoint	Inkorrekte Darstell-ung der Bildnummer
1	–	–	–	–	–
2	–	–	–	–	–
3	–	–	–	–	–
4	–	–	–	–	–
5	–	–	–	–	–
6	–	–	–	–	–
7	–	–	–	–	–
8	–	–	–	–	–
9	–	–	–	–	–
10	–	–	–	–	–
11	–	–	–	–	–
12	–	–	–	–	–
13	–	–	–	–	–
14	–	–	–	–	–
15	–	–	–	–	–
16	–	–	–	–	–
17	–	–	–	–	–
18	–	–	–	–	–
19	–	–	–	–	–
20	–	–	–	–	–
21	–	–	–	–	–
22	–	–	–	–	–
23	–	–	–	–	–
24	–	–	–	–	–
25	–	–	–	–	–
26	–	–	–	–	–
27	–	–	–	–	–
28	–	–	–	–	–
29	–	–	–	–	–
30	–	–	–	–	–
31	–	–	–	–	–
32	–	–	–	–	–
33	–	–	–	–	–
34	–	–	–	–	–
35	–	–	–	–	–
Sequence-Header korrekt verwendet				ok	
Sequence-End-Code korrekt verwendet				ok	

Tabelle C.2.: Testprotokoll nach Punkt 2: Test aller Schnittmöglichkeiten bei Interlaced Framepictures UFF

Szene	Bildsortier-fehler	Encodier-fehler	Farb-fehler	Inkorrekter In- oder Outpoint	Inkorrekte Darstell-ung der Bildnummer
1	–	–	–	–	–
2	–	–	–	–	–
3	–	–	–	–	–
4	–	–	–	–	–
5	–	–	–	–	–
6	–	–	–	–	–
7	–	–	–	–	–
8	–	–	–	–	–
9	–	–	–	–	–
10	–	–	–	–	–
11	–	–	–	–	–
12	–	–	–	–	–
13	–	–	–	–	–
14	–	–	–	–	–
15	–	–	–	–	–
16	–	–	–	–	–
17	–	–	–	–	–
18	–	–	–	–	–
19	–	–	–	–	–
20	–	–	–	–	–
21	–	–	–	–	–
22	–	–	–	–	–
23	–	–	–	–	–
24	–	–	–	–	–
25	–	–	–	–	–
26	–	–	–	–	–
27	–	–	–	–	–
28	–	–	–	–	–
29	–	–	–	–	–
30	–	–	–	–	–
31	–	–	–	–	–
32	–	–	–	–	–
33	–	–	–	–	–
34	–	–	–	–	–
35	–	–	–	–	–
Sequence-Header korrekt verwendet				ok	
Sequence-End-Code korrekt verwendet				ok	

Tabelle C.3.: Testprotokoll nach Punkt 3: Test aller Schnittmöglichkeiten bei Interlaced Fieldpictures UFF

Szene	Bildsortier-fehler	Encodier-fehler	Farb-fehler	Inkorrekter In- oder Outpoint	Inkorrekte Darstell-ung der Bildnummer
1	–	–	–	–	–
2	–	–	–	–	–
3	–	–	–	–	–
4	–	–	–	–	–
5	–	–	–	–	–
6	–	–	–	–	–
7	–	–	–	–	–
8	–	–	–	–	–
9	–	–	–	–	–
10	–	–	–	–	–
11	–	–	–	–	–
12	–	–	–	–	–
13	–	–	–	–	–
14	–	–	–	–	–
15	–	–	–	–	–
16	–	–	–	–	–
17	–	–	–	–	–
18	–	–	–	–	–
19	–	–	–	–	–
20	–	–	–	–	–
21	–	–	–	–	–
22	–	–	–	–	–
23	–	–	–	–	–
24	–	–	–	–	–
25	–	–	–	–	–
26	–	–	–	–	–
27	–	–	–	–	–
28	–	–	–	–	–
29	–	–	–	–	–
Sequence-Header korrekt verwendet				ok	
Sequence-End-Code korrekt verwendet				ok	

Tabelle C.4.: Testprotokoll nach Punkt 4: Test aller Schnittmöglichkeiten mit langer *GOP*-Struktur

Szene	Bildsortier-fehler	Farbfehler	Inkorrekte Darstellung der Bildnummer
1	–	–	–
2	–	–	–
3	–	–	–
4	–	–	–
5	–	–	–
Sequence-Header korrekt verwendet			ok
Sequence-End-Code korrekt verwendet			ok

Tabelle C.5.: Testprotokoll nach Punkt 5: Test aller Schnittmöglichkeiten mit dem Intraframe-Video

In die EDL eingebaute Fehler	Reaktion der Software
die EDL ist nicht verfügbar	ok
ein Dateiname ist falsch geschrieben	ok
ein Inpoint ist größer als der dazugehörige Outpoint	ok
es wird ein Link benutzt, welcher größer als die Anzahl der gesamt verfügbaren Links ist	ok
es wird die Datei `_hphl_422.m2v` gemeinsam mit `_[farbe]_prog.m2v` eingetragen	ok
es wird die Datei `_[farbe]_frame_bot.m2v` gemeinsam mit `_[farbe]_frame_top.m2v` eingetragen	ok
es wird die Datei `_hires_video.m2v` zusammen mit `_[farbe]_field_bot.m2v` eingetragen	ok

Tabelle C.6.: Testprotokoll nach Punkt 6: Test der Ausnahmefälle